NETCONF Server and RESTCONF Server Configuration Models
draft-ietf-netconf-server-model-07

Abstract

This draft defines a NETCONF server configuration data model and a
RESTCONF server configuration data model. These data models enable
configuration of the NETCONF and RESTCONF services themselves,
including which transports are supported, what ports the servers
listen on, call-home parameters, client authentication, and other
related configuration parameters.

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.

This document contains references to other drafts in progress, both
in the Normative References section, as well as in body text
throughout. Please update the following references to reflect their
final RFC assignments:

- draft-ietf-netconf-restconf
- draft-ietf-netconf-call-home

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

- "VVVV" --> the assigned RFC value for this draft
- "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-call-home
- "ZZZZ" --> the assigned RFC value for draft-thomson-httpbis-cant
Artwork in this document contains placeholder values for ports pending IANA assignment from "draft-ietf-netconf-call-home". Please apply the following replacements:

- "7777" --> the assigned port value for "netconf-ch-ssh"
- "8888" --> the assigned port value for "netconf-ch-tls"
- "9999" --> the assigned port value for "restconf-ch-tls"

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- "2015-07-06" --> the publication date of this draft

The following two Appendix sections are to be removed prior to publication:

- Appendix B. Change Log
- Appendix C. Open Issues

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 7, 2016.

Copyright Notice

Copyright (c) 2015 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
Table of Contents

1. Introduction ................................................. 4
   1.1. Terminology ........................................... 4
   1.2. Tree Diagrams ......................................... 5
2. Objectives ................................................ 5
   2.1. Support all NETCONF and RESTCONF transports ........... 5
   2.2. Enable each transport to select which keys to use ....... 5
   2.3. Support authenticating NETCONF/RESTCONF clients ......... 6
   2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames ............... 6
   2.5. Support both listening for connections and call home .... 6
   2.6. For Call Home connections ................................ 6
       2.6.1. Support more than one NETCONF/RESTCONF client .... 6
       2.6.2. Support NETCONF/RESTCONF clients having more than one endpoint ......................... 6
       2.6.3. Support a reconnection strategy ..................... 7
       2.6.4. Support both persistent and periodic connections ... 7
       2.6.5. Reconnection strategy for periodic connections ... 7
       2.6.6. Keep-ales for persistent connections ............... 7
       2.6.7. Customizations for periodic connections ........... 8
3. The NETCONF Server Model ..................................... 8
   3.1. Tree Diagram ........................................... 8
   3.2. Example Usage .......................................... 9
       3.2.1. Configuring SSH Transport ......................... 10
       3.2.2. Configuring TLS Transport ......................... 11
   3.3. YANG Model .............................................. 13
4. The RESTCONF Server Model .................................... 26
   4.1. Tree Diagram ........................................... 26
   4.2. Example Usage .......................................... 27
       4.2.1. Configuring TLS Transport ......................... 27
   4.3. YANG Model .............................................. 28
5. Security Considerations ....................................... 37
6. IANA Considerations .......................................... 38
7. Other Considerations ......................................... 39
8. Acknowledgements ........................................... 39
9. References .................................................. 39
   9.1. Normative References .................................... 39
   9.2. Informative References .................................. 40
Appendix A. Alternative solution addressing Issue #49 ............ 41
   A.1. The Keychain Model ..................................... 41
1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model and a RESTCONF [draft-ietf-netconf-restconf] server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, call-home parameters, client authentication, and other related configuration parameters.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].
1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

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

2. Objectives

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF and RESTCONF services on a network element. This scope includes the following objectives:

2.1. Support all NETCONF and RESTCONF transports

The YANG module should support all current NETCONF and RESTCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [RFC7589], and RESTCONF over TLS [draft-ietf-netconf-restconf], and to be extensible to support future transports as necessary.

Because implementations may not support all transports, the module should use YANG "feature" statements so that implementations can accurately advertise which transports are supported.

2.2. Enable each transport to select which keys to use

Servers may have a multiplicity of host-keys or server-certificates from which subsets may be selected for specific uses. For instance, a NETCONF server may want to use one set of SSH host-keys when listening on port 830, and a different set of SSH host-keys when calling home. The data models provided herein should enable configuration of which keys to use on a per-use basis.
2.3. Support authenticating NETCONF/RESTCONF clients certificates

When a certificate is used to authenticate a NETCONF or RESTCONF client, there is a need to configure the server to know how to authenticate the certificates. The server should be able to authenticate the client’s certificate either by using path-validation to a configured trust anchor or by matching the client-certificate to one previously configured.

2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames

When a client certificate is used for TLS client authentication, the NETCONF/RESTCONF server must be able to derive a username from the authenticated certificate. Thus the modules defined herein should enable this mapping to be configured.

2.5. Support both listening for connections and call home

The NETCONF and RESTCONF protocols were originally defined as having the server opening a port to listen for client connections. More recently the NETCONF working group defined support for call-home ([draft-ietf-netconf-call-home]), enabling the server to initiate the connection to the client, for both the NETCONF and RESTCONF protocols. Thus the modules defined herein should enable configuration for both listening for connections and calling home. Because implementations may not support both listening for connections and calling home, YANG "feature" statements should be used so that implementation can accurately advertise the connection types it supports.

2.6. For Call Home connections

The following objectives only pertain to call home connections.

2.6.1. Support more than one NETCONF/RESTCONF client

A NETCONF/RESTCONF server may be managed by more than one NETCONF/RESTCONF client. For instance, a deployment may have one client for provisioning and another for fault monitoring. Therefore, when it is desired for a server to initiate call home connections, it should be able to do so to more than one client.

2.6.2. Support NETCONF/RESTCONF clients having more than one endpoint

An NETCONF/RESTCONF client managing a NETCONF/RESTCONF server may implement a high-availability strategy employing a multiplicity of active and/or passive endpoint. Therefore, when it is desired for a
server to initiate call home connections, it should be able to connect to any of the client’s endpoints.

2.6.3. Support a reconnection strategy

Assuming a NETCONF/RESTCONF client has more than one endpoint, then it becomes necessary to configure how a NETCONF/RESTCONF server should reconnect to the client should it lose its connection to one of the client’s endpoints. For instance, the NETCONF/RESTCONF server may start with first endpoint defined in a user-ordered list of endpoints or with the last endpoints it was connected to.

2.6.4. Support both persistent and periodic connections

NETCONF/RESTCONF clients may vary greatly on how frequently they need to interact with a NETCONF/RESTCONF server, how responsive interactions need to be, and how many simultaneous connections they can support. Some clients may need a persistent connection to servers to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for server to initiate connections, it should be configurable if the connection is persistent or periodic.

2.6.5. Reconnection strategy for periodic connections

The reconnection strategy should apply to both persistent and periodic connections. How it applies to periodic connections becomes clear when considering that a periodic "connection" is a logical connection to a single server. That is, the periods of unconnectedness are intentional as opposed to due to external reasons. A periodic "connection" should always reconnect to the same server until it is no longer able to, at which time the reconnection strategy guides how to connect to another server.

2.6.6. Keep-alives for persistent connections

If a persistent connection is desired, it is the responsibility of the connection initiator to actively test the "aliveness" of the connection. The connection initiator must immediately work to reestablish a persistent connection as soon as the connection is lost. How often the connection should be tested is driven by NETCONF/RESTCONF client requirements, and therefore keep-alive settings should be configurable on a per-client basis.
2.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF/RESTCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF/RESTCONF client may have to wait to send data to a server. A server may connect to a client before this interval expires if desired (e.g., to send data to a client).

3. The NETCONF Server Model

3.1. Tree Diagram

module: ietf-netconf-server
   +--rw netconf-server
      +--rw session-options
         | +--rw hello-timeout? uint16
         +--rw listen { (ssh-listen or tls-listen) }?
            | +--rw max-sessions? uint16
            +--rw idle-timeout? uint16
            +--rw endpoint* [name]
               +--rw name string
               +--rw (transport)
               | +--:(ssh) { ssh-listen }?
               |    +--rw ssh
               |    | +--rw address? inet:ip-address
               |    | +--rw port? inet:port-number
               |    +--rw host-keys
               |    +--rw host-key* string
               | +--:(tls) { tls-listen }?
               |    +--rw tls
               |    | +--rw address? inet:ip-address
               |    | +--rw port? inet:port-number
               |    +--rw certificates
               |    +--rw certificate* string
            +--rw call-home { ( ssh-call-home or tls-call-home ) }?
               +--rw netconf-client* [name]
               | +--rw name string
               +--rw (transport)
               | +--:(ssh) { ssh-call-home }?
               |    +--rw ssh
               |    | +--rw endpoints
               |    | | +--rw endpoint* [name]
               |    |    | +--rw name string
               |    |    | +--rw address inet:ip-address
               |    |    | +--rw port? inet:port-number
               |    |    +--rw host-keys
               |    |    +--rw host-key* string
               | +--:(tls) { tls-call-home }?
3.2. Example Usage
3.2.1. Configuring SSH Transport

The following example illustrates the <get> response from a NETCONF server that only supports SSH, both listening for incoming connections as well as calling home to a single NETCONF/RESTCONF client having two endpoints.

```xml
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <listen>
    <endpoint>
      <name>netconf/ssh</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>my-rsa-key</host-key>
          <host-key>my-dss-key</host-key>
        </host-keys>
      </ssh>
    </endpoint>
  </listen>
  <call-home>
    <netconf-client>
      <name>config-mgr</name>
      <ssh>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>55.66.77.88</address>
          </endpoint>
        </endpoints>
        <host-keys>
          <host-key>my-call-home-x509-key</host-key>
        </host-keys>
      </ssh>
    </netconf-client>
  </call-home>
</netconf-server>
```
3.2.2. Configuring TLS Transport

The following example illustrates the <get> response from a NETCONF server that only supports TLS, both listening for incoming connections as well as calling home to a single NETCONF/RESTCONF client having two endpoints. Please note also the configurations for authenticating client certificates and mappings authenticated certificates to NETCONF user names.

```
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server"
                xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </endpoint>
  </listen>

  <call-home>
    <netconf-client>
      <name>config-mgr</name>
      <tls>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>22.33.44.55</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>33.44.55.66</address>
          </endpoint>
        </endpoints>
      </tls>
    </netconf-client>
  </call-home>
</netconf-server>
```
<certificate>IDEvID Certificate</certificate>
</certificates>
</tls>
</call-home>
</netconf-client>
</trusted-ca-certs>

WmdsK2gyTtG3qmtGMjhWbW1cdFFFFaWc3OEgrRkYyRTFwdSt42ZRJbVFFM1lQiJsdWpocjFTMnRLR05EMU20VwpK2FNGWNG2NTd2NctadVJ1M2pRYjzkSfNwSdWxVXBCyEnumFWdNtJma3JqzHXeFppUUtTbndWZTF2Zwot
NGcEk3UE90cnNFvjrwTUNBd0VbQWFPQ0FSXdnZ0VFck1CMEdBMVVkgD
VEJiJ20JTiWd1UEKMrhpRHvOTvkvVHLWd4cfFJBz120UyUOEzRz05ER
V6qUJc205WqkFvNEIt1NquO0KJyZNOMVyorYSONDUUVRHBS116U8gZREF
NQmdOvksSTU2B2jhFckFqGUNFtRHTqFVZ3RUIvd1FFQxdJSGdEONbC
Z050SF14RvlqMnrKzJsnXFBB2hoNW9kSFJ3T2k4dpYaGgYK1hCci1tPN
WpiMb2W1hoagjQjQnNaUzVqY215aU9LUTJNRFF43pBSkJnT12CQV1UQW
QmdOvkJBWVrBfZUTVJbad0RnWURUWFLRXdbAp1R020Y0d4bE1RNdEoQ
MkF6a3hqueDIWQWtHR0dvSlU1ueUc1SVR0Wm0vK3B0R2FieXVMDjBRd2kvz
25PznpZNEbONpXy07aUp2K2xtyWxs3RTORUZX9S9Rdp4NUL1XmdvN2RJ5UJQFtS0Cg==
</trusted-client-certs>
</trusted-client-certs>

QmdOvkJBWVrBfZUTVJbad0RnWURUWFLRXdbAp1R020Y0d4bE1RNdEoQ
MkF6a3hqueDIWQWtHR0dvSlU1ueUc1SVR0Wm0vK3B0R2FieXVMDjBRd2kvz
25PznpZNEbONpXy07aUp2K2xtyWxs3RTORUZX9S9Rdp4NUL1XmdvN2RJ5UJQFtS0Cg==

QmdOvkJBWVrBfZUTVJbad0RnWURUWFLRXdbAp1R020Y0d4bE1RNdEoQ
MkF6a3hqueDIWQWtHR0dvSlU1ueUc1SVR0Wm0vK3B0R2FieXVMDjBRd2kvz
25PznpZNEbONpXy07aUp2K2xtyWxs3RTORUZX9S9Rdp4NUL1XmdvN2RJ5UJQFtS0Cg==
<trusted-client-cert>
</trusted-client-certs>
<cert-maps>
  <cert-to-name>
    <id>1</id>
    <fingerprint>11:0A:05:11:00</fingerprint>
    <map-type>x509c2n:sn-any</map-type>
  </cert-to-name>
  <cert-to-name>
    <id>2</id>
    <fingerprint>B3:4F:A1:8C:54</fingerprint>
    <map-type>x509c2n:specified</map-type>
    <name>scooby-doo</name>
  </cert-to-name>
</cert-maps>
</client-auth>
</tls>
</netconf-server>

3.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-netconf-server@2015-07-06.yang"

module ietf-netconf-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
  prefix "ncserver";

  import ietf-netconf-acm {
    prefix nacm; // RFC 6536
  }
  import ietf-inet-types { // RFC 6991
```
Watsen & Schoenwaelder   Expires January 7, 2016   [Page 13]`
prefix inet;
}
import ietf-x509-cert-to-name { // RFC 7407
    prefix x509c2n;
}

organization
"IETF NETCONF (Network Configuration) Working Group";

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

WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nsn.com>

WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com>

Editor: Kent Watsen
<mailto:kwatsen@juniper.net>";

description
"This module contains a collection of YANG definitions for configuring NETCONF servers.

Copyright (c) 2014 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 VVVV; see the RFC itself for full legal notices.";

revision "2015-07-06" {
    description
    "Initial version";
    reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}
// Features

feature ssh-listen {
  description "The ssh-listen feature indicates that the NETCONF server supports opening a port to accept NETCONF over SSH client connections.";
  reference "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature ssh-call-home {
  description "The ssh-call-home feature indicates that the NETCONF server supports initiating a NETCONF over SSH call home connection to NETCONF clients.";
  reference "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
  description "The tls-listen feature indicates that the NETCONF server supports opening a port to accept NETCONF over TLS client connections.";
  reference "RFC 5539: Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication";
}

feature tls-call-home {
  description "The tls-call-home feature indicates that the NETCONF server supports initiating a NETCONF over TLS call home connection to NETCONF clients.";
  reference "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature ssh-x509-certs {
  description "The ssh-x509-certs feature indicates that the NETCONF server supports RFC 6187";
  reference "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}
// top-level container  (groupings below)
container netconf-server {
  description
    "Top-level container for NETCONF server configuration.";

  container session-options {  // SHOULD WE REMOVE THIS ALTOGETHER?
    description
      "NETCONF session options, independent of transport
       or connection strategy.";
    leaf hello-timeout {
      type uint16;
      units "seconds";
      default 600;
      description
        "Specifies the maximum number of seconds that a SSH/TLS
         connection may wait for a hello message to be received.
         A connection will be dropped if no hello message is
         received before this number of seconds elapses. If set
to zero, then the server will wait forever for a hello
         message.";
    }
  }

  container listen {
    description
      "Configures listen behavior";
    if-feature "(ssh-listen or tls-listen)";
    leaf max-sessions {
      type uint16;
      default 0;
      description
        "Specifies the maximum number of concurrent sessions
         that can be active at one time. The value 0 indicates
         that no artificial session limit should be used.";
    }

    leaf idle-timeout {
      type uint16;
      units "seconds";
      default 3600; // one hour
      description
        "Specifies the maximum number of seconds that a NETCONF
         session may remain idle. A NETCONF session will be dropped
         if it is idle for an interval longer than this number of
         seconds. If set to zero, then the server will never drop
         a session because it is idle. Sessions that have a
         notification subscription active are never dropped.";
    }

    list endpoint {

key name;

description
"List of endpoints to listen for NETCONF connections on."

leaf name {
    type string;
    description
    "An arbitrary name for the NETCONF listen endpoint."
}

choice transport {
    mandatory true;
    description
    "Selects between available transports."

    case ssh {
        if-feature ssh-listen;
        container ssh {
            description
            "SSH-specific listening configuration for inbound
             connections.";
            uses address-and-port-grouping {
                refine port {
                    default 830;
                }
            }
            uses host-keys-grouping;
        }
    }

    case tls {
        if-feature tls-listen;
        container tls {
            description
            "TLS-specific listening configuration for inbound
             connections.";
            uses address-and-port-grouping {
                refine port {
                    default 6513;
                }
            }
            uses certificates-grouping;
        }
    }
}

container call-home {
    if-feature "((ssh-call-home or tls-call-home))"
    description
    "Configures call-home behavior";
list netconf-client {
  key name;
  description
    "List of NETCONF clients the NETCONF server is to initiate
call-home connections to."
  leaf name {
    type string;
    description
      "An arbitrary name for the remote NETCONF client.";
  }
  choice transport {
    mandatory true;
    description
      "Selects between available transports.";
    case ssh {
      if-feature ssh-call-home;
      container ssh {
        description
          "Specifies SSH-specific call-home transport
configuration."
        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 7777;
          }
        }
        uses host-keys-grouping;
      }
    }
    case tls {
      if-feature tls-call-home;
      container tls {
        description
          "Specifies TLS-specific call-home transport
configuration."
        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 8888;
          }
        }
        uses certificates-grouping;
      }
    }
  }
  container connection-type {
    description
      "Indicates the kind of connection to use.";
    choice connection-type {
      description
        "Indicates the kind of connection to use.";
    }
  }
}
"Selects between available connection types."

case persistent-connection {
    container persistent {
        presence true;
        description
            "Maintain a persistent connection to the NETCONF client. If the connection goes down, immediately start trying to reconnect to it, using the reconnection strategy.

This connection type minimizes any NETCONF client to NETCONF server data-transfer delay, albeit at the expense of holding resources longer."

    leaf idle-timeout {
        type uint32;
        units "seconds";
        default 86400;  // one day;
        description
            "Specifies the maximum number of seconds that a NETCONF session may remain idle. A NETCONF session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
    }
    container keep-alives {
        description
            "Configures the keep-alive policy, to proactively test the aliveness of the SSH/TLS client. An unresponsive SSH/TLS client will be dropped after approximately (max-attempts * max-wait) seconds."
        reference
            "RFC YYYY: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6";
    leaf max-wait {
        type uint16 {
            range "1..max";
        }
        units seconds;
        default 30;
        description
            "Sets the amount of time in seconds after which if no data has been received from the SSH/TLS client, a SSH/TLS-level message will be sent to test the aliveness of the SSH/TLS client.";
    }
}
leaf max-attempts {
    type uint8;
    default 3;
    description
    "Sets the number of sequential keep-alive messages
    that can fail to obtain a response from the SSH/TLS
    client before assuming the SSH/TLS client is no
    longer alive.";
}
}
}
case periodic-connection {
    container periodic {
        presence true;
        description
        "Periodically connect to the NETCONF client, so that
        the NETCONF client may deliver messages pending for
        the NETCONF server. The NETCONF client is expected
to close the connection when it is ready to release
it, thus starting the NETCONF server’s timer until
next connection.";
    leaf idle-timeout {
        type uint16;
        units "seconds";
        default 300; // five minutes
        description
        "Specifies the maximum number of seconds that a
        NETCONF session may remain idle. A NETCONF
        session will be dropped if it is idle for an
        interval longer than this number of seconds.
        If set to zero, then the server will never drop
        a session because it is idle. Sessions that
        have a notification subscription active are
        never dropped.";
    }
    leaf reconnect_timeout {
        type uint16 {
            range "1..max";
        }
        units minutes;
        default 60;
        description
        "The maximum amount of unconnected time the NETCONF
        server will wait before establishing a connection
        to the NETCONF client. The NETCONF server may
        initiate a connection before this time if desired
        (e.g., to deliver a notification).";
container reconnect-strategy {
    description
    "The reconnection strategy guides how a NETCONF server reconnects to an NETCONF client, after losing a connection to it, even if due to a reboot. The NETCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin)."

    leaf start-with {
        type enumeration {
            enum first-listed {
                description
                "Indicates that reconnections should start with the first endpoint listed.";
            }
            enum last-connected {
                description
                "Indicates that reconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. NETCONF servers SHOULD be able to remember the last endpoint connected to across reboots.";
            }
        }
        default first-listed;
        description
        "Specifies which of the NETCONF client’s endpoints the NETCONF server should start with when trying to connect to the NETCONF client.";
    }

    leaf max-attempts {
        type uint8 {
            range "1..max";
        }
        default 3;
        description
        "Specifies the number times the NETCONF server tries to connect to a specific endpoint before moving on to the next endpoint in the list (round robin).";
    }
}
container ssh {
  description
  "Configures SSH properties not specific to the listen
  or call-home use-cases";
  if-feature "(ssh-listen or ssh-call-home)"
  container x509 {
    if-feature ssh-x509-certs;
    uses trusted-certs-grouping;
  }
}

container tls {
  description
  "Configures TLS properties for authenticating clients.";
  if-feature "(tls-listen or tls-call-home)"
  container client-auth {
    description
    "Container for TLS client authentication configuration.";
    uses trusted-certs-grouping;
  container cert-maps {
    uses x509c2n:cert-to-name;
    description
    "The cert-maps container is used by a NETCONF server to
    map the NETCONF client’s presented X.509 certificate to a
    NETCONF username. If no matching and valid cert-to-name
    list entry can be found, then the NETCONF server MUST
    close the connection, and MUST NOT accept NETCONF
    messages over it.";
    reference
    "RFC WWWW: NETCONF over TLS, Section 7";
  }
}
}

grouping trusted-certs-grouping {
  description
  "This grouping is used by both the ssh and tls containers.";
  container trusted-ca-certs {
    description
    "A list of Certificate Authority (CA) certificates that
    a NETCONF server can use to authenticate NETCONF client
    certificates.";
    reference
    "RFC WWWW: NETCONF over TLS, Sections 5 and 7.
    RFC 4253: The Secure Shell (SSH) Transport Layer Protocol,
leaf-list trusted-ca-cert {
  type binary;
  nacm:default-deny-write;
  description
    "The binary certificate structure as specified by RFC
    5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
    ";
  reference
    "RFC 5246: The Transport Layer Security (TLS)
    Protocol Version 1.2";
}
}

container trusted-client-certs {
  description
    "A list of client certificates that a NETCONF server can
    use to authenticate a NETCONF client’s certificate. A
    client’s certificate is authenticated if it is an exact
    match to a configured trusted client certificate.";
  leaf-list trusted-client-cert {
    type binary;
    nacm:default-deny-write;
    description
      "The binary certificate structure, as
      specified by RFC 5246, Section 7.4.6, i.e.,:
      
      opaque ASN.1Cert<1..2^24>;
      
      ";
    reference
      "RFC 5246: The Transport Layer Security (TLS)
      Protocol Version 1.2";
  }
}
}

grouping host-keys-grouping {
  description
    "This grouping is used by both the listen and
call-home containers";
  container host-keys {
    description
      "Parent container for the list of host-keys.";
    leaf-list host-key {
      type string;
      min-elements 1;
  }
}
ordered-by user;

description
"A user-ordered list of host-keys the SSH server considers when composing the list of server host key algorithms it will send to the client in its SSH_MSG_KEXINIT message. The value of the string is the unique identifier for a host-key configured on the system. How valid values are discovered is outside the scope of this module, but they are envisioned to be the keys for a list of host-keys provided by another YANG module";

reference
"RFC 4253: The SSH Transport Layer Protocol, Section 7";

}
}


grouping certificates-grouping {

description
"This grouping is used by both the listen and call-home containers";

container certificates {

description
"Parent container for the list of certificates.";

leaf-list certificate {

type string;

min-elements 1;

description
"An unordered list of certificates the TLS server can pick from when sending its Server Certificate message. The value of the string is the unique identifier for a certificate configured on the system. How valid values are discovered is outside the scope of this module, but they are envisioned to be the keys for a list of certificates provided by another YANG module";

reference
"RFC 5246: The TLS Protocol, Section 7.4.2";

}
}


grouping address-and-port-grouping {

description
"This grouping is used by both the ssh and tls containers for listen configuration.";

leaf address {

type inet:ip-address;

description
"The IP address of the interface to listen on. The NETCONF server will listen on all interfaces if no value is specified."

leaf port {
  type inet:port-number;
  description
  "The local port number on this interface the NETCONF server listens on. The NETCONF server will use the IANA-assigned well-known port if no value is specified.";
}
}
}

grouping endpoints-container {
  description
  "This grouping is used by both the ssh and tls containers for call-home configurations.";
  container endpoints {
    description
    "Container for the list of endpoints.";
    list endpoint {
      key name;
      min-elements 1;
      ordered-by user;
      description
      "User-ordered list of endpoints for this NETCONF client. Defining more than one enables high-availability.";
      leaf name {
        type string;
        description
        "An arbitrary name for this endpoint.";
      }
      leaf address {
        type inet:host;
        mandatory true;
        description
        "The IP address or hostname of the endpoint. If a hostname is configured and the DNS resolution results in more than one IP address, the NETCONF server will process the IP addresses as if they had been explicitly configured in place of the hostname.";
      }
      leaf port {
        type inet:port-number;
        description
        "The IP port for this endpoint. The NETCONF server will use the IANA-assigned well-known port if no value is
4. The RESTCONF Server Model

4.1. Tree Diagram

module: ietf-restconf-server
  +--rw restconf-server
    +--rw listen (tls-listen)?
      |  +--rw max-sessions?  uint16
      |  +--rw endpoint* [name]
      |    +--rw name  string
      |    +--rw (transport)
      |    |  +--:(tls)
      |    |   +--rw tls
      |    |   |  +--rw address?  inet:ip-address
      |    |   |  +--rw port?  inet:port-number
      |    |   |  +--rw certificates
      |    |   |   +--rw certificate*  string
      |  +--rw call-home (tls-call-home)?
      |  +--rw restconf-client* [name]
      |    +--rw name  string
      |    +--rw (transport)
      |    |  +--:(tls)
      |    |   +--rw tls
      |    |   |  +--rw endpoints
      |    |   |   |  +--rw endpoint* [name]
      |    |   |   |    +--rw name  string
      |    |   |   |    +--rw address  inet:host
      |    |   |   |    +--rw port?  inet:port-number
      |    |   |   |   +--rw certificates
      |    |   |   |   +--rw certificate*  string
      |    +--rw connection-type
      |    |  +--rw (connection-type)?
      |    |    +--:(persistent-connection)
      |    |    |  +--rw persistent!
      |    |    |  +--rw keep-alives
      |    |    |     +--rw max-wait?  uint16
      |    |    |     +--rw max-attempts?  uint8
4.2. Example Usage

4.2.1. Configuring TLS Transport

The following example illustrates the <get> response from a RESTCONF server that only supports TLS, both listening for incoming connections as well as calling home to a single RESTCONF client having two endpoints.
<restconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">
  <listen>
    <endpoint>
      <name>primary-restconf-endpoint</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </endpoint>
  </listen>
  <call-home>
    <restconf-client>
      <name>config-mgr</name>
      <tls>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>55.66.77.88</address>
          </endpoint>
        </endpoints>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </restconf-client>
  </call-home>
</restconf-server>

4.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].
prefix nacm; // RFC 6536
}
import ietf-inet-types { // RFC 6991
    prefix inet;
}
import ietf-x509-cert-to-name { // RFC 7407
    prefix x509c2n;
}

organization
"IETF NETCONF (Network Configuration) Working Group";

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

WG Chair: Mehmet Ersue
        <mailto:mehmet.ersue@nsn.com>

WG Chair: Mahesh Jethanandani
        <mailto:mjethanandani@gmail.com>

Editor:  Kent Watsen
        <mailto:kwatsen@juniper.net>";

description
"This module contains a collection of YANG definitions for
configuring RESTCONF servers.

Copyright (c) 2014 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 VVVV; see
the RFC itself for full legal notices.";

revision "2015-07-06" {
    description
        "Initial version";
    reference
        "RFC VVVV: NETCONF Server and RESTCONF Server Configuration";
// Features

feature tls-listen {
  description
  "The listen feature indicates that the RESTCONF server supports opening a port to listen for incoming RESTCONF client connections.";
  reference
  "RFC XXXX: RESTCONF Protocol";
}

feature tls-call-home {
  description
  "The call-home feature indicates that the RESTCONF server supports initiating connections to RESTCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature client-cert-auth {
  description
  "The client-cert-auth feature indicates that the RESTCONF server supports the ClientCertificate authentication scheme.";
  reference
  "RFC ZZZZ: Client Authentication over New TLS Connection";
}

// top-level container  (groupings below)
container restconf-server {
  description
  "Top-level container for RESTCONF server configuration.";
}

container listen {
  description
  "Configures listen behavior";
  if-feature tls-listen;
  leaf max-sessions {
    type uint16;
    default 0;  // should this be 'max'? 
    description
    "Specifies the maximum number of concurrent sessions that can be active at one time. The value 0 indicates that no artificial session limit should be used.";
  }
}
list endpoint {
    key name;
    description
        "List of endpoints to listen for RESTCONF connections on.";
    leaf name {
        type string;
        description
            "An arbitrary name for the RESTCONF listen endpoint."
    }
}

choice transport {
    mandatory true;
    description
        "Selects between available transports.";
    case tls {
        container tls {
            description
                "TLS-specific listening configuration for inbound
                connections.";
            leaf address {
                type inet:ip-address;
                description
                    "The IP address of the interface to listen on. The
                    RESTCONF server will listen on all interfaces if
                    no value is specified.";
            }
            leaf port {
                type inet:port-number;
                default 443;
                description
                    "The port number the RESTCONF server will listen on.";
            }
        }
    }
}

container call-home {
    if-feature tls-call-home;
    description
        "Configures call-home behavior";
    list restconf-client {
        key name;
        description
            "List of RESTCONF clients the RESTCONF server is to
            initiate call-home connections to.";
    }
}
leaf name {
    type string;
    description
    "An arbitrary name for the remote RESTCONF client.";
}
choice transport {
    mandatory true;
    description
    "Selects between TLS and any transports augmented in.";
    case tls {
        container tls {
            description
            "Specifies TLS-specific call-home transport configuration.";
        }
        container endpoints {
            description
            "Container for the list of endpoints.";
        }
        list endpoint {
            key name;
            min-elements 1;
            ordered-by user;
            description
            "User-ordered list of endpoints for this RESTCONF client. More than one enables high-availability.";
            leaf name {
                type string;
                description
                "An arbitrary name for this endpoint.";
            }
            leaf address {
                type inet:host;
                mandatory true;
                description
                "The IP address or hostname of the endpoint. If a hostname is configured and the DNS resolution results in more than one IP address, the RESTCONF server will process the IP addresses as if they had been explicitly configured in place of the hostname.";
            }
            leaf port {
                type inet:port-number;
                default 9999;
                description
                "The IP port for this endpoint. The RESTCONF server will use the IANA-assigned well-known port if no value is specified.";
            }
        }
    }
}
container connection-type {
    description
    "Indicates the RESTCONF client’s preference for how the
    RESTCONF server’s connection is maintained.";
    choice connection-type {
        description
        "Selects between available connection types.";
        case persistent-connection {
            container persistent {
                presence true;
                description
                "Maintain a persistent connection to the RESTCONF
                client. If the connection goes down, immediately
                start trying to reconnect to it, using the
                reconnection strategy.
                
                This connection type minimizes any RESTCONF client
to RESTCONF server data-transfer delay, albeit at
the expense of holding resources longer.";
            }
        }
    }
    container keep-alives {
        description
        "Configures the keep-alive policy, to proactively
test the aliveness of the TLS client. An
unresponsive TLS client will be dropped after
approximately (max-attempts * max-wait) seconds.";
        reference
        "RFC YYYY: NETCONF Call Home and RESTCONF Call Home,
        Section 3.1, item S6";
        leaf max-wait {
            type uint16 {
                range "1..max";
            }
            units seconds;
            default 30;
            description
            "Sets the amount of time in seconds after which
if no data has been received from the TLS
client, a TLS-level message will be sent to
test the aliveness of the TLS client.";
        }
        leaf max-attempts {
        }
    }
}
type uint8;
default 3;
description
"Sets the number of sequential keep-alive messages
that can fail to obtain a response from the TLS
client before assuming the TLS client is no
longer alive."
}
}
}
}
}
{
case periodic-connection {
    container periodic {
        presence true;
description
"Periodically connect to the RESTCONF client, so that
the RESTCONF client may deliver messages pending for
the RESTCONF server. The RESTCONF client is expected
to close the connection when it is ready to release
it, thus starting the RESTCONF server's timer until
next connection."
leaf reconnect-timeout {
    type uint16 {
        range "1..max";
    }
    units minutes;
default 60;
description
"The maximum amount of unconnected time the RESTCONF
server will wait before re-establishing a connection
to the RESTCONF client. The RESTCONF server may
initiate a connection before this time if desired
(e.g., to deliver a notification)."
}
}
}
}
{
case reconnect-strategy {
    container reconnect-strategy {
description
"The reconnection strategy guides how a RESTCONF server
reconnects to an RESTCONF client, after losing a connection
to it, even if due to a reboot. The RESTCONF server starts
with the specified endpoint and tries to connect to it
max-attempts times before trying the next endpoint in the
list (round robin)."
leaf start-with {
    type enumeration {
enum first-listed {
  description
  "Indicates that reconnections should start with
  the first endpoint listed.";
}
enum last-connected {
  description
  "Indicates that reconnections should start with
  the endpoint last connected to. If no previous
  connection has ever been established, then the
  first endpoint configured is used. RESTCONF
  servers SHOULD be able to remember the last
  endpoint connected to across reboots.";
}
default first-listed;
description
  "Specifies which of the RESTCONF client’s endpoints the
  RESTCONF server should start with when trying to connect
  to the RESTCONF client.";
}
leaf max-attempts {
  type uint8 {
    range "1..max";
  }
default 3;
description
  "Specifies the number times the RESTCONF server tries to
  connect to a specific endpoint before moving on to the
  next endpoint in the list (round robin).";
}
}

container client-cert-auth {
  if-feature client-cert-auth;
description
  "Container for TLS client certificate authentication
  configuration.";
  container trusted-ca-certs {
    description
    "A list of Certificate Authority (CA) certificates that
    a RESTCONF server can use to authenticate RESTCONF client
    certificates.";
    reference
    "RFC XXXX: RESTCONF Protocol, Sections 2.3 and 2.5.";
    leaf-list trusted-ca-cert {
container trusted-client-certs {
  description
  "A list of client certificates that a RESTCONF server can
  use to authenticate a RESTCONF client’s certificate. A
  client’s certificate is authenticated if it is an exact
  match to a configured trusted client certificate."
  leaf-list trusted-client-cert {
    type binary;
    nacm:default-deny-write;
    description
    "The binary certificate structure, as specified by RFC
    5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
    ";
    reference
    "RFC 5246: The Transport Layer Security (TLS)
    Protocol Version 1.2";
  }
}

certificates-grouping {
  description
  "This grouping is used by both the listen and
call-home containers;
container certificates {
   description  "Parent container for the list of certificates.";
   leaf-list certificate {
      type string;
      min-elements 1;
      description  "An unordered list of certificates the TLS server can pick
from when sending its Server Certificate message. The value
of the string is the unique identifier for a certificate
configured on the system. How valid values are discovered
is outside the scope of this module, but they are envisioned
to be the keys for a list of certificates provided
by another YANG module";
      reference  "RFC 5246: The TLS Protocol, Section 7.4.2";
   }
}
}

5. Security Considerations

There are a number of data nodes defined in the "ietf-netconf-server"
YANG module which are readable and/or writable that may be considered
sensitive or vulnerable in some network environments. Write and read
operations to these data nodes can have a negative effect on network
operations. It is thus important to control write and read access to
these data nodes. Below are the data nodes and their sensitivity/
vulnerability.

netconf-server/tls/client-auth/trusted-ca-certs:

o This container contains certificates that a NETCONF server is to
use as trust anchors for authenticating X.509-based client
certificates. Write access to this node is protected using an
nacm:default-deny-write statement.

netconf-server/tls/client-auth/trusted-client-certs:

o This container contains certificates that a NETCONF server is to
trust directly when authenticating X.509-based client
certificates. Write access to this node is protected using an nacm:default-deny-write statement.

restconf-server/tls/client-auth/trusted-ca-certs:

- This container contains certificates that a RESTCONF server is to use as trust anchors for authenticating X.509-based client certificates. Write access to this node is protected using an nacm:default-deny-write statement.

restconf-server/tls/client-auth/trusted-client-certs:

- This container contains certificates that a RESTCONF server is to trust directly when authenticating X.509-based client certificates. Write access to this node is protected using an nacm:default-deny-write statement.

6. IANA Considerations

This document registers two URIs in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registrations are requested:

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

name:         ietf-netconf-server
prefix:       ncserver
reference:    RFC VVVV

name:         ietf-restconf-server
prefix:       rcserver
reference:    RFC VVVV
7. Other Considerations

The YANG modules define herein do not themselves support virtual routing and forwarding (VRF). It is expected that external modules will augment in VRF designations when needed.

8. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, and Bert Wijnen.

Juergen Schoenwaelder and was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

9. References

9.1. Normative References


9.2. Informative References

Appendix A. Alternative solution addressing Issue #49

Option #4 for Issue #49 proposed to define configuration for a keychain and on-going discussion proposed to create reusable groupings for SSH/TLS servers (referencing keys and certificates held in the keychain) that the NETCONF/RESTCONF servers would uses. This relationship is illustrated by the diagram below.

```
+-------------+
|ietf-keychain|
+-------------+
    ^    ^
   <leafref> | | <leafref>
      +--------+ +--------+
     |     |     |     |
    |ielf-ssh-server| ielf-tls-server |
    +--------+ +--------+
         ^       ^
       <uses> | <augments> |
         +--------+ +--------+
     |ielf-netconf-server| ielf-restconf-server|
     +--------+ +--------+
```

The following sections each of the five YANG modules above.

A.1. The Keychain Model

A.1.1. Tree Diagram
module: ietf-keychain

---rw keychain
  ---rw private-keys
    ---rw private-key* [name]
      ---rw name string
      ---ro algorithm? enumeration
      ---ro key-length? uint32
      ---ro public-key? string
      ---rw certificates
        ---rw certificate* [name]
          ---rw name string
          ---rw chain? binary
    ---rw trusted-certificates* [name]
      ---rw name string
      ---rw trusted-certificate* [name]
        ---rw name string
        ---rw certificate? binary

rpcs:
  ---x generate-certificate-signing-request
    ---w input
      ---w private-key? -> /keychain/private-keys/private-key/name
      ---w subject binary
      ---w attributes? binary
    ---ro output
      ---ro certificate-signing-request binary
  ---x generate-private-key
    ---w input
      ---w name string
      ---w algorithm enumeration
      ---w key-length uint32

A.1.2. Example Usage

<kyc

<!-- private keys and associated certificates -->
<private-keys>
  <private-key>
    <name>TPM key</name>
    <algorithm>rsa</algorithm>
    <key-length>2048</key-length>
    <public-key>
      cztvaWRo2RmZ2tqaHKnkZmdramRzZnZzZGtmacm5idnNvO2RmanZvO3NkZmJpdmhzZGpHejdv21kZmhidmlibHKnkYmZ2aXN1ZGpYmhzZG87zmJvO3NkZ25iO29pLmR6Zgo=
    </public-key>
  </private-key>
  <certificates>
  </certificates>

Watsen & Schoenwaelder Expires January 7, 2016 [Page 42]
<name>IDevID Certificate</name>

<chain>

LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z0F3SUJBZ01KQUpRT2t3bGpNK2pjTUEwRwNTcUdTSWizRFFQkJRUVFNRRFQ4Q3pBSkJnT1YQkFZVEF5vV1RUkF3RgdZFRZRUUtFd2RsZUdGdGNHeGxNUk13RgRFZ2RUUyRd3BEVWt3Z1NYTnpkV1Z5TU10ApeDiR1V4RXpBUkJnT1ZCQ1UQ2tOU1RDQkpmjM04xWiJdJ2daOHdEUU1KS29aSWh2Y04QKVFFQkJRQURnWTBBTU1HskFvR0BKTXVzVFNEV3E1lQWMzR1RsTkNmc0d6cEw1Um5ydXZsOFRlicUJTDGZQY3N0Zk1KT1FNaN1N1NlVdsm1dzaHE1bEUU1CiJKJNniGNaUdbTAuU25FcFE0TnVbXBDT2YQkWdNQkFBR2pnYXzD2Fr0hRwURWUjBPQkJZRUZKY1o2WUIr0IPNDB4ajj1b3tjRTEdsRUNCVTFNR1FHQTFVZApDj1JkTU1QUZKYL1o2WUJR01PNDB4ajj1b3tjRTEdsRUNCVTFvGGl1rTmBEMi1rc3dmKUkUE0ExeVWREd3RUR53UNCREFTQmdOVkhSTUJBZjhFQ0RBFR0B29ZBOVTEwRwNTcUdTSWizRFFQgppCUVQBQTHQkFMXnrWmFGNWcyGR6MWNhZnZ2PbBnNEA4eG00SRRhBStadjPpLazFL1SBxTXp4YXJcbPDS1hLck1Vbc9GVzRtV1Qs1VDeEFT40NEYYZm2kd4c0dtSSElkyYW1LWpGMlQ5OSVXFT4K11aDZmazcrQzQ1QXg1RWWSWhgzjdVM2xZTGotLS0tLUVORCBDRVJUSUZQ0FUR0tLS0tCg==
</chain>

</certificates>

</private-key>
</private-keys>

<!-- trusted netconf/restconf client certificates -->

<trusted-certificates>

<name>Trusted certificates for netconf/restconf client</name>

<trusted-certificate>

<name>George Jetson</name>

<certificate>

QmdOVkB6WVRbfZUTVJBd0RnWURUVF0RvRxbApl1R0Z0Y0d4bE1RNHDdEQMKF6a3hQgUlDQVWtR0dsv1UeUC1SVRoWmV3B0R2FieXVDMjBRd2kvZ25PZnpZNEhONApAYXy0pAUpZK2xtYW35RTRORUZXS9rdp4NULXZmdvN2RVOJCUt2MNXFNeUFUVPswMyY0WtXbUUNEOakJrQmdOVkhTTVEUYJeJiZ0JWTDedUBeUMnhpRHoTVKvVHFLNd4cFJBZ12OYUU0cERzD05ERUxNQWhtQFTVUJoUWNVwi4URB0TJnT1ZCQ9UjJWNAPA2V3sF3yKdeVEV6VQJCZ05QKFNFVENzT1UNQJQKYzNOMPVYcSNDUUNVRHBSn1616GBzREFNQmdOVkhSTUJBzhFckFqUQNFTHRQTFVZER3UR1viDFFQXDJSgcdENBCZ05WSF41ZvLqMndMNzJnSXFBZ2hO9NkSFJZTdk4dlpYa2hyKhtC1pTnpWpImJb2W1holKJQYQykNaUzVqY215aUIUTJNRF4Q3pBSkJnT1ZCQV1QWxWWE15QXdEz11EVRZSpFvDz2UbGdGNHeGxNUk13RFZ2FU2UyRD3BEVWt3Z1NYTnpkV1Z5TU10ApeDiR1V4RXpBUkJnT1ZCQ1UQ2tOU1RDQkpmjM04xWiJdJ2daOHdEUU1KS29aSWh2Y04QKVFFQkJRQURnWTBBTU1HskFvR0BKTXVzVFNEV3E1lQWMzR1RsTkNmc0d6cEw1Um5ydXZsOFRlicUJTDGZQY3N0Zk1KT1FNaN1N1NlVdsm1dzaHE1bEUU1CiJKJNniGNaUdbTAuU25FcFE0TnVbXBDT2YQkWdNQkFBR2pnYXzD2Fr0hRwURWUjBPQkJZRUZKY1o2WUIr0IPNDB4ajj1b3tjRTEdsRUNCVTFNR1FHQTFVZApDj1JkTU1QUZKYL1o2WUJR01PNDB4ajj1b3tjRTEdsRUNCVTFvGGl1rTmBEMi1rc3dmKUkUE0ExeVWREd3RUR53UNCREFTQmdOVkhSTUJBZjhFQ0RBFR0B29ZBOVTEwRwNTcUdTSWizRFFQgppCUVQBQTHQkFMXnrWmFGNWcyGR6MWNhZnZ2PbBnNEA4eG00SRRhBStadjPpLazFL1SBxTXp4YXJcbPDS1hLck1Vbc9GVzRtV1Qs1VDeEFT40NEYYZm2kd4c0dtSSElkyYW1LWpGMlQ5OSVXFT4K11aDZmazcrQzQ1QXg1RWWSWhgzjdVM2xZTGotLS0tLUVORCBDRVJUSUZQ0FUR0tLS0tCg==
</certificate>

</trusted-certificate>

</trusted-certificates>
</trusted-certificate>
</trusted-certificates>
</trusted-certificates>
</trusted-certificates>

<!-- trust anchors for netconf/restconf clients -->
<trusted-certificates>
  <name>Trust anchors for netconf/restconf clients</name>
</trusted-certificates>
</trusted-certificates>
</trusted-certificates>

<!-- trust anchors for random HTTPS servers on Internet -->
<trusted-certificates>
</trusted-certificates>
<name>Trust anchors for random HTTPS servers</name>
<trusted-certificate>
<name>Example.com</name>
<certificate>
NGcEk3UE90cnNFVjRwTUNBd0VBQWPQ0FSSXdn20VPCk1CMEmBMVVkRGdVEJiZ0JTE6dlBEUEMnhpRHVOTVkvVHFLNw4cFJB21ZOUUU0cERZd05ERV6QVJCZ05WQkFNVENrT1NUQ0JJKYzNOMVpYSONDUNVRHBN5116UG8zREFZ05WSFt4RVI1QmdNRj2nSXFBZ2hoNW9kSFJ3T2k4d1pYaGgY1hCc1pTNQmdOVkJBWVRBbFZUTV JBdOrnWURWUWVFIRXdbkAp1R020Y04bEIRNHDdEQMKf6a3hQUD1VQWtHR0dvS1U1eUc1SVROw0vK3B0R2FieXVDMjBRd2kvZ
NQmdOVkshTUJBZjhFCkFqQUFNQTRHQTFRVZER3RUIvd1FFQxdJSgcdE0mBCWmdsK2gyTTg3Qm tGMJhWBw1CdFVawc3OEgrRkYyRTFwdSt42VRJbVF FMMLQ1lsdWpOcjFTMnRLO5EMUc20VJpK2FWNGw2NTdZNCtadVJM2gPYj k zSFNwSdSdwVXBCYnA4dmtNanFtZjma3RqZHBeFppUUtTbdWZTF2Zwot25Pzn5P2EhNnApxY0tpaUpZK5xtyW35TRORUZXZ9RdGp4NU1XZmdvZ
WpMjB2W1hoaGJQYnNaUzVqVqY215aU9L=
</certificate>
</trusted-certificate>
</trusted-certificates>
</keychain>

A.1.3. YANG Model

<CODE BEGINS> file "ietf-keychain@2015-07-06.yang"

module ietf-keychain {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keychain";
  prefix "kc";

  organization
  "IETF NETCONF (Network Configuration) Working Group";

  contact
  "WG Web:  <http://tools.ietf.org/wg/netconf/>
  WG List:  <mailto:netconf@ietf.org>
  WG Chair: Mehmet Ersue
            <mailto:mehmet.ersue@nsn.com>
  WG Chair: Mahesh Jethanandani
            <mailto:mjethanandani@gmail.com>
  Editor:   Kent Watsen
            <mailto:kwatsen@juniper.net>";

</CODE ENDS>
description
"This module defines a keychain to centralize management of
security credentials.

Copyright (c) 2014 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 VVVV; see
the RFC itself for full legal notices.";

revision "2015-07-06" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
     Models";
}

container keychain {
  description
    "A list of private-keys and their associated certificates, as
     well as lists of trusted certificates for client certificate
     authentication. RPCs are provided to generate a new private
     key and to generate a certificate signing requests.";
}

container private-keys {
  description
    "A list of private key maintained by the keychain.";
  list private-key {
    key name;
    description
      "A private key.";
    leaf name {
      type string;
      description
        "An arbitrary name for the private key.";
    }
    leaf algorithm {
      type enumeration {
        enum rsa { description "TBD"; }
        enum dsa { description "TBD"; }
      }
    }
  }
}
enum secp192r1 { description "TBD"; }
enum sect163k1 { description "TBD"; }
enum sect163r2 { description "TBD"; }
enum secp224r1 { description "TBD"; }
enum sect233k1 { description "TBD"; }
enum sect233r1 { description "TBD"; }
enum secp256r1 { description "TBD"; }
enum sect283k1 { description "TBD"; }
enum sect283r1 { description "TBD"; }
enum secp384r1 { description "TBD"; }
enum sect409k1 { description "TBD"; }
enum sect409r1 { description "TBD"; }
enum secp521r1 { description "TBD"; }
enum sect571k1 { description "TBD"; }
enum sect571r1 { description "TBD"; }
}
config false;
description
"The algorithm used by the private key."
;
}
leaf key-length {
type uint32;
config false;
description
"The key-length used by the private key.";
}
leaf public-key {
type string;
config false;
description
"The public-key matching the private key.";
}
container certificates {
list certificate {
key name;
description
"A certificate for this public key.";
leaf name {
type string;
description
"An arbitrary name for the certificate.";
}
leaf chain {
type binary;
description
"The certificate itself, as well as an ordered sequence of intermediate certificates leading to a trust anchor, as specified by RFC 5246,

Section 7.4.2.

action generate-certificate-signing-request {
  description
  "Generates a certificate signing request structure for
  the associated private key using the passed subject
  and attribute values.";
  input {
    leaf subject {
      type binary;
      mandatory true;
      description
      "The 'subject' field in the CertificationRequestInfo
       defined in RFC 2986, Section 4.1.";
      reference
      "RFC 2986: PKCS #10: Certification Request Syntax
       Specification Version 1.7";
    }
    leaf attributes {
      type binary;
      description
      "The 'attributes' field in the CertificationRequestInfo
       defined in RFC 2986, Section 4.1.";
      reference
      "RFC 2986: PKCS #10: Certification Request Syntax
       Specification Version 1.7";
    }
  }
  output {
    leaf certificate-signing-request {
      type binary;
      mandatory true;
      description
      "The CertificationRequestInfo structure as specified
       by RFC 2986, Section 4.1.";
      reference
      "RFC 2986: PKCS #10: Certification Request Syntax
       Specification Version 1.7";
    }
  }
}
action generate-private-key {
  description
  "Generates a private key using the specified algorithm and
  key length.";
  input {
    leaf name {
      type string;
      mandatory true;
      description
      "The name this private-key should have when listed in
       /keychain/private-keys/private-key. As such, the
       passed value must not match any existing 'name' value.";
    }
    leaf algorithm {
      type enumeration {
        enum rsa { description "TBD"; }
        enum dsa { description "TBD"; }
        enum secp192r1 { description "TBD"; }
        enum sect163k1 { description "TBD"; }
        enum sect163r2 { description "TBD"; }
        enum secp224r1 { description "TBD"; }
        enum sect233k1 { description "TBD"; }
        enum sect233r1 { description "TBD"; }
        enum secp256r1 { description "TBD"; }
        enum sect283k1 { description "TBD"; }
        enum sect283r1 { description "TBD"; }
        enum secp384r1 { description "TBD"; }
        enum sect409k1 { description "TBD"; }
        enum sect409r1 { description "TBD"; }
        enum secp521r1 { description "TBD"; }
        enum sect571k1 { description "TBD"; }
        enum sect571r1 { description "TBD"; }
      }
      mandatory true;
      description
      "The algorithm to be used.";
    }
    leaf key-length {
      type uint32;
      mandatory true;
      description
      "The key length to be used.";
    }
  }
}
list trusted-certificates {
    key name;
    description
        "A list of lists of trusted certificates.";
    leaf name {
        type string;
        description
            "An arbitrary name for this list of trusted certificates.";
    }
}

list trusted-certificate {
    key name;
    description
        "A list of trusted certificates for a specific use.";
    leaf name {
        type string;
        description
            "An arbitrary name for this trusted certificate.";
    }
    leaf certificate {
        type binary;
        description
            "The binary certificate structure as specified by RFC 5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;",
        reference
    }
}

rpc generate-certificate-signing-request {
    description
        "Generates a certificate signing request structure for the specified private key using the passed subject and attribute values.";
    input {
        leaf private-key {
            type leafref {
                path "[/keychain/private-keys/private-key/name];";
            }
            description
                "The private key to generate the certificate signing request for.";
        }
        leaf subject {
            type binary;
        }
    }
}
manditory true;
description
"The 'subject' field in the CertificationRequestInfo
defined in RFC 2986, Section 4.1."
reference
"RFC 2986: PKCS #10: Certification Request Syntax
Specification Version 1.7";
}
leaf attributes {
type binary;
description
"The 'attributes' field in the CertificationRequestInfo
defined in RFC 2986, Section 4.1."
reference
"RFC 2986: PKCS #10: Certification Request Syntax
Specification Version 1.7";
}
output {
leaf certificate-signing-request {
type binary;
manditory true;
description
"The CertificationRequestInfo structure as specified
by RFC 2986, Section 4.1."
reference
"RFC 2986: PKCS #10: Certification Request Syntax
Specification Version 1.7";
}
}

call generate-private-key {
description
"Generates a private key using the specified algorithm and
key length."
input {
leaf name {
type string;
manditory true;
description
"The name this private-key should have when listed in
/keychain/private-keys/private-key. As such, the
passed value must not match any existing 'name' value."
}
leaf algorithm {
type enumeration {
enum rsa { description "TBD"; }
}
}
A.2. The SSH Server Model

A.2.1. Tree Diagram

The following tree diagram is faked, as a module having only a grouping in it has no tree diagram. However, for illustrative purposes, a container has been added as nothing more than a "uses" statement of the grouping.
A.2.2. Example Usage

```
<fake-ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">
  <host-keys>
    <host-key>
      <name>IDevID</name>
      <certificate>
        IDevID Certificate
      </certificate>
    </host-key>
  </host-keys>
  <client-cert-auth>
    <trusted-ca-certs>
      Trusted certificates for netconf/restconf clients
    </trusted-ca-certs>
    <trusted-client-certs>
      Trust anchors for netconf/restconf clients
    </trusted-client-certs>
  </client-cert-auth>
</fake-ssh-server>
```

A.2.3. YANG Model

```
<CODE BEGINS> file "ietf-ssh-server@2015-07-06.yang"

module ietf-ssh-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
  prefix "ts";
}
```

Watsen & Schoennwaelder Expires January 7, 2016 [Page 53]
import ietf-keychain {
    prefix kc;                      // RFC VVVV
}

organization
"IETF NETCONF (Network Configuration) Working Group";

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

WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nsn.com>

WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com>

Editor: Kent Watsen
<mailto:kwatsen@juniper.net>";

description
"This module defines a reusable grouping for a SSH server that
can be used as a basis for specific SSH server instances.

Copyright (c) 2014 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 VVVV; see
the RFC itself for full legal notices.";

revision "2015-07-06" {
    description
        "Initial version";
    reference
        "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
        Models";
}

// features
feature ssh-x509-certs {
description
  "The ssh-x509-certs feature indicates that the NETCONF server
  supports RFC 6187";
reference
  "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}

// grouping
grouping ssh-server-grouping {
  description
    "A reusable grouping for a SSH server that can be used as a
    basis for specific SSH server instances."
  container host-keys {
    description
      "The list of host-keys the SSH server will present when
      establishing a SSH connection.";
    list host-key {
      key name;
      min-elements 1;
      ordered-by user;
      description
        "An ordered list of hostkeys the SSH server advertises
        when sending its Authentication message.";
      reference
        "RFC 3160: ...";
      leaf name {
        type string;
        mandatory true;
        description
          "An arbitrary name for this host-key";
      }
      choice type {
        leaf public-key {
          type leafref {
            path "#/keychain/keychain/private-keys/private-key/private-key/name";
          }
          description
            "The name of a private-key in the keychain.";
        }
        leaf certificate {
          if-feature ssh-x509-certs;
          type leafref {
            path "#/keychain/keychain/private-keys/private-key/certificates/certificate/certificate/name";
          }
          description
            "The name of a certificate in the keychain.";
        }
      }
  }
}
container client-cert-auth {
  if-feature ssh-x509-certs;
  description
    "A reference to a list of trusted certificate authority (CA) certificates and a reference to a list of trusted client certificates.";
  leaf trusted-ca-certs {
    type leafref {
      path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
    description
      "A reference to a list of certificate authority (CA) certificates used by the SSH server to authenticate SSH client certificates.";
  }
  leaf trusted-client-certs {
    type leafref {
      path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
    description
      "A reference to a list of client certificates used by the SSH server to authenticate SSH client certificates. A clients certificate is authenticated if it is an exact match to a configured trusted client certificate.";
  }
}

A.3. The TLS Server Model

A.3.1. Tree Diagram

The following tree diagram is faked, as a module having only a grouping in it has no tree diagram. However, for illustrative purposes, a container has been added as nothing more than a "uses" statement of the grouping.
module: ietf-tls-server
  +--rw fake-tls-server
    +--rw certificates
      |   +--rw certificate* [name]
      |      +--rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
      +--rw client-auth
        +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
        +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name

A.3.2. Example Usage

<fake-tls-server xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <certificates>
    <certificate>
      IDevID Certificate
    </certificate>
  </certificates>
  <client-auth>
    <trusted-ca-certs>
      Trusted certificates for netconf/restconf clients
    </trusted-ca-certs>
    <trusted-client-certs>
      Trust anchors for netconf/restconf clients
    </trusted-client-certs>
  </client-auth>
</fake-tls-server>

A.3.3. YANG Model

<CODE BEGINS> file "ietf-tls-server@2015-07-06.yang"

module ietf-tls-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
  prefix "ts";

  import ietf-keychain {
    prefix kc;  // RFC VVVV
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/netconf/>

Watsen & Schoenwaelder   Expires January 7, 2016   [Page 57]
description
"This module defines a reusable grouping for a TLS server that can be used as a basis for specific TLS server instances.

Copyright (c) 2014 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 VVVV; see the RFC itself for full legal notices."

revision "2015-07-06" {
  description "Initial version";
  reference "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

grouping tls-server-grouping {
  description "A reusable grouping for a TLS server that can be used as a basis for specific TLS server instances.";

  container certificates {
    description "The list of certificates the TLS server will present when establishing a TLS connection.";
    list certificate {

key name;
min-elements 1;
description
"An unordered list of certificates the TLS server can pick
from when sending its Server Certificate message.";
reference
"RFC 5246: The TLS Protocol, Section 7.4.2";
leaf name {
  type leafref {
    path */kc:keychain/kc:private-keys/kc:private-key/kc:certificates/kc:certificate/kc:name*;
  }
description
"The name of the certificate in the keychain.";
}
}
container client-auth {
  description
  "A reference to a list of trusted certificate authority (CA)
certificates and a reference to a list of trusted client
certificates.";
  leaf trusted-ca-certs {
    type leafref {
      path */kc:keychain/kc:trusted-certificates/kc:name*;
    }
description
    "A reference to a list of certificate authority (CA)
certificates used by the TLS server to authenticate
TLS client certificates.";
  }
  leaf trusted-client-certs {
    type leafref {
      path */kc:keychain/kc:trusted-certificates/kc:name*;
    }
description
    "A reference to a list of client certificates used by
the TLS server to authenticate TLS client certificates.
A client certificate is authenticated if it is an
exact match to a configured trusted client certificate.";
  }
}
}
A.4. The NETCONF Server Model

A.4.1. Tree Diagram

module: ietf-netconf-server-new

---rw netconf-server
  +--rw session-options
  |  +--rw hello-timeout? uint16
  |  +--rw max-sessions? uint16
  +--rw idle-timeout? uint16
  +--rw name string
  +--rw (transport)
  |  +--:(ssh) (ssh-listen)?
  |     +--rw ssh
  |        +--rw address? inet:ip-address
  |        +--rw port? inet:port-number
  |        +--rw host-keys
  |            +--rw host-key* [name]
  |            |  +--rw name string
  |            |  +--rw (type)?
  |            |     +--:(public-key)
  |            |        +--rw public-key? -> /kc:keychain/private-keys/private-key/name
  |            |        +--:(certificate)
  |            |            +--rw certificate? -> /kc:keychain/private-keys/private-key/certificates/certificate/name {ssh-x509-certs}?
  |            +--rw client-cert-auth {ssh-x509-certs}?
  |            +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
  |        +--:(tls) (tls-listen)?
  |            +--rw tls
  |                +--rw address? inet:ip-address
  |                +--rw port? inet:port-number
  |                +--rw certificates
  |                    +--rw certificate* [name]
  |                    |  +--rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
  |                    +--rw trusted-cert-auth
  |                        +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
  |                        +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
  |                        +--rw cert-maps
  |                        |  +--rw cert-to-name* [id]
  |                        |     +--rw id uint32
  |                        |     +--rw fingerprint x509c2n:tls-fingerprint
  |                        |     +--rw map-type identityref
  |                        +--rw name string
  +--rw call-home {(ssh-call-home or tls-call-home)?

Watsen & Schoenwaelder Expires January 7, 2016 [Page 60]
A.4.2.  Example Usage

Configuring an SSH Server

<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <listen>
    <endpoint>
      <name>netconf/ssh</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>
            <public-key>my-rsa-key</public-key>
          </host-key>
        </host-keys>
        <client-cert-auth>
          <trusted-ca-certs>
            Trusted netconf/restconf client certificates
          </trusted-ca-certs>
          <trusted-client-certs>
            Trust anchors for netconf/restconf clients
          </trusted-client-certs>
        </client-cert-auth>
      </ssh>
    </endpoint>
  </listen>
  <call-home>
    <netconf-client>
      <name>config-mgr</name>
      <ssh>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
        </endpoints>
      </ssh>
    </netconf-client>
  </call-home>
</netconf-server>
Configuring a TLS Server

  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>IDevID Certificate</certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>
            Trusted netconf/restconf client certificates
          </trusted-ca-certs>
          <trusted-client-certs>
            Trust anchors for netconf/restconf clients
          </trusted-client-certs>
          <cert-maps>
            <cert-to-name>
              <id>1</id>
              <fingerprint>11:0A:05:11:00</fingerprint>
              <map-type>x509c2n:san-any</map-type>
            </cert-to-name>
            <cert-to-name>
              <id>2</id>
            </cert-to-name>
          </cert-maps>
        </client-auth>
      </tls>
    </endpoint>
  </listen>
</netconf-server>
A.4.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-netconf-server-new@2015-07-06.yang"

module ietf-netconf-server-new {
  yang-version 1.1;

  prefix "ncserver";

  import ietf-inet-types { // RFC 6991
    prefix inet;
  }

Watsen & Schoenwaelder   Expires January 7, 2016               [Page 64]
import ietf-x509-cert-to-name { // RFC 7407
  prefix x509c2n;
}
import ietf-ssh-server { // RFC VVVV
  prefix ss;
}
import ietf-tls-server { // RFC VVVV
  prefix ts;
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
  "WG Web: <http://tools.ietf.org/wg/netconf/>
  WG List: <mailto:netconf@ietf.org>
  WG Chair: Mehmet Ersue
    <mailto:mehmet.ersue@nsn.com>
  WG Chair: Mahesh Jethanandani
    <mailto:mjethanandani@gmail.com>
  Editor: Kent Watsen
    <mailto:kwatsen@juniper.net>"

description
  "This module contains a collection of YANG definitions for
  configuring NETCONF servers.

  Copyright (c) 2014 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 VVVV; see
  the RFC itself for full legal notices.";

revision "2015-07-06" {
  description
    "Initial version";
feature ssh-listen {
  description
  "The ssh-listen feature indicates that the NETCONF server
  supports opening a port to accept NETCONF over SSH
  client connections.";
  reference
  "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature ssh-call-home {
  description
  "The ssh-call-home feature indicates that the NETCONF server
  supports initiating a NETCONF over SSH call
  home connection to NETCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
  description
  "The tls-listen feature indicates that the NETCONF server
  supports opening a port to accept NETCONF over TLS
  client connections.";
  reference
  "RFC 5539: Using the NETCONF Protocol over Transport
  Layer Security (TLS) with Mutual X.509 Authentication";
}

feature tls-call-home {
  description
  "The tls-call-home feature indicates that the NETCONF server
  supports initiating a NETCONF over TLS call
  home connection to NETCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature ssh-x509-certs {
  description
  "The ssh-x509-certs feature indicates that the NETCONF server
  supports using X.509 certificates for
  authentication";
  reference
  "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
  Models";
}

// Features
"The ssh-x509-certs feature indicates that the NETCONF server supports RFC 6187";
reference "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";

// top-level container  (groupings below)
container netconf-server {
  description "Top-level container for NETCONF server configuration.";

  container session-options { // SHOULD WE REMOVE THIS ALTOGETHER?
    description "NETCONF session options, independent of transport or connection strategy.";
    leaf hello-timeout {
      type uint16;
      units "seconds";
      default 600;
      description "Specifies the maximum number of seconds that a SSH/TLS connection may wait for a hello message to be received. A connection will be dropped if no hello message is received before this number of seconds elapses. If set to zero, then the server will wait forever for a hello message.";
    }
  }

  container listen {
    description "Configures listen behavior";
    if-feature "(ssh-listen or tls-listen)";
    leaf max-sessions {
      type uint16;
      default 0;
      description "Specifies the maximum number of concurrent sessions that can be active at one time. The value 0 indicates that no artificial session limit should be used.";
    }

    leaf idle-timeout {
      type uint16;
      units "seconds";
      default 3600; // one hour
      description "Specifies the maximum number of seconds that a NETCONF
session may remain idle. A NETCONF session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.

```yaml
list endpoint {
  key name;
  description  "List of endpoints to listen for NETCONF connections on.";
  leaf name {
    type string;
    description  "An arbitrary name for the NETCONF listen endpoint.";
  }
  choice transport {
    mandatory true;
    description  "Selects between available transports.";
    case ssh {
      if-feature ssh-listen;
      container ssh {
        description  "SSH-specific listening configuration for inbound connections.";
        uses address-and-port-grouping {
          refine port {
            default 830;
          }
        }
        uses ss:ssh-server-grouping;
      }
    }
    case tls {
      if-feature tls-listen;
      container tls {
        description  "TLS-specific listening configuration for inbound connections.";
        uses address-and-port-grouping {
          refine port {
            default 6513;
          }
        }
        uses tls-server-grouping;
      }
    }
  }
}
```
container call-home {
  if-feature "(ssh-call-home or tls-call-home)";
  description "Configures call-home behavior";
  list netconf-client {
    key name;
    description "List of NETCONF clients the NETCONF server is to initiate call-home connections to.";
    leaf name {
      type string;
      description "An arbitrary name for the remote NETCONF client.";
    }
    choice transport {
      mandatory true;
      description "Selects between available transports.";
      case ssh {
        if-feature ssh-call-home;
        container ssh {
          description "Specifies SSH-specific call-home transport configuration.";
          uses endpoints-container {
            refine endpoints/endpoint/port {
              default 7777;
            }
          }
          uses ss:ssh-server-grouping;
        }
      }
      case tls {
        if-feature tls-call-home;
        container tls {
          description "Specifies TLS-specific call-home transport configuration.";
          uses endpoints-container {
            refine endpoints/endpoint/port {
              default 8888;
            }
          }
          uses tls-server-grouping;
        }
      }
    }
  }
}
container connection-type {
    description "Indicates the kind of connection to use.";
    choice connection-type {
        description "Selects between available connection types.";
        case persistent-connection {
            container persistent {
                presence true;
                description "Maintain a persistent connection to the NETCONF client. If the connection goes down, immediately start trying to reconnect to it, using the reconnection strategy.

This connection type minimizes any NETCONF client to NETCONF server data-transfer delay, albeit at the expense of holding resources longer.";
            leaf idle-timeout {
                type uint32;
                units "seconds";
                default 86400;  // one day;
                description "Specifies the maximum number of seconds that a NETCONF session may remain idle. A NETCONF session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
            }
        }
        case keep-alives {
            container keep-alives {
                description "Configures the keep-alive policy, to proactively test the aliveness of the SSH/TLS client. An unresponsive SSH/TLS client will be dropped after approximately (max-attempts * max-wait) seconds.";
                reference "RFC YYYY: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6";
                leaf max-wait {
                    type uint16 {
                        range "1..max";
                    }
                    units seconds;
                }
            }
        }
    }
}


default 30;

description
"Sets the amount of time in seconds after which
if no data has been received from the SSH/TLS
client, a SSH/TLS-level message will be sent
to test the aliveness of the SSH/TLS client."
);
leaf max-attempts {
  type uint8;
  default 3;
  description
  "Sets the number of sequential keep-alive messages
  that can fail to obtain a response from the SSH/TLS
  client before assuming the SSH/TLS client is no
  longer alive.";
}

case periodic-connection {
  container periodic {
    presence true;
    description
    "Periodically connect to the NETCONF client, so that
    the NETCONF client may deliver messages pending for
    the NETCONF server. The NETCONF client is expected
to close the connection when it is ready to release
it, thus starting the NETCONF server’s timer until
next connection."
leaf idle-timeout {
  type uint16;
  units "seconds";
  default 300; // five minutes
  description
  "Specifies the maximum number of seconds that a
  NETCONF session may remain idle. A NETCONF
  session will be dropped if it is idle for an
  interval longer than this number of seconds.
  If set to zero, then the server will never drop
  a session because it is idle. Sessions that
  have a notification subscription active are
  never dropped.";

  leaf reconnect_timeout {
    type uint16 {
      range "1..max";
    }
    units minutes;
container reconnect-strategy {
  description
  "The reconnection strategy guides how a NETCONF server reconnects to an NETCONF client, after losing a connection to it, even if due to a reboot. The NETCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";
  leaf start-with {
    type enumeration {
      enum first-listed {
        description
        "Indicates that reconnections should start with the first endpoint listed.";
      }
      enum last-connected {
        description
        "Indicates that reconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. NETCONF servers SHOULD be able to remember the last endpoint connected to across reboots.";
      }
      default first-listed;
    }
  }
  leaf max-attempts {
    type uint8 {
      range "1..max";
    }
    default 3;
}

default 60;

description
"The maximum amount of unconnected time the NETCONF server will wait before re-establishing a connection to the NETCONF client. The NETCONF server may initiate a connection before this time if desired (e.g., to deliver a notification).";
description
"Specifies the number times the NETCONF server tries to
connect to a specific endpoint before moving on to the
next endpoint in the list (round robin).";
}
}
}
}


grouping tls-server-grouping {
  description
  "An augmentation of tls-server-grouping, as defined in the
  ietf-tls-server module, to add in cert-maps."
  uses ts:tls-server-grouping {
    augment "client-auth" {
      container cert-maps {
        uses x509c2n:cert-to-name;
        description
        "The cert-maps container is used by a NETCONF server to
        map the NETCONF client’s presented X.509 certificate to a
        NETCONF username. If no matching and valid cert-to-name
        list entry can be found, then the NETCONF server MUST
        close the connection, and MUST NOT accept NETCONF
        messages over it."
      }
      reference
      "RFC WWWW: NETCONF over TLS, Section 7";
    }
  }
}


grouping address-and-port-grouping {
  description
  "This grouping is used by both the ssh and tls containers
  for listen configuration."
  leaf address {
    type inet:ip-address;
    description
    "The IP address of the interface to listen on. The NETCONF
    server will listen on all interfaces if no value is
    specified."
  }
  leaf port {
    type inet:port-number;
    description
    "The local port number on this interface the NETCONF server
listens on. The NETCONF server will use the IANA-assigned well-known port if no value is specified.

grouping endpoints-container {
    description
        "This grouping is used by both the ssh and tls containers for call-home configurations.",
    container endpoints {
        description
            "Container for the list of endpoints.",
        list endpoint {
            key name;
            min-elements 1;
            ordered-by user;
            description
                "User-ordered list of endpoints for this NETCONF client. Defining more than one enables high-availability.",
            leaf name {
                type string;
                description
                    "An arbitrary name for this endpoint.",
            }
            leaf address {
                type inet:host;
                mandatory true;
                description
                    "The IP address or hostname of the endpoint. If a hostname is configured and the DNS resolution results in more than one IP address, the NETCONF server will process the IP addresses as if they had been explicitly configured in place of the hostname.",
            }
            leaf port {
                type inet:port-number;
                description
                    "The IP port for this endpoint. The NETCONF server will use the IANA-assigned well-known port if no value is specified.",
            }
        }
    }
}
A.5. The RESTCONF Server Model

A.5.1. Tree Diagram

module: ietf-restconf-server-new
  +--rw restconf-server
    |  +--rw listen {tls-listen}?
    |     |  +--rw max-sessions? uint16
    |     +--rw endpoint* [name]
    |     |  +--rw name string
    |     +--rw (transport)
    |        +--:(tls)
    |            +--rw address? inet:ip-address
    |            +--rw port? inet:port-number
    |            +--rw certificates
    |            |  +--rw certificate* [name]
    |            |     |  +--rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
    |            |     +--rw client-auth
    |            |     +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
    |            |     +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
    |            |     +--rw cert-maps
    |            |     +--rw cert-to-name* [id]
    |            |            +--rw id uint32
    |            |            +--rw fingerprint x509c2n:tls-fingerprint
    |            +--rw name string
    +--rw call-home {tls-call-home}?
    +--rw restconf-client* [name]
    |  +--rw name string
    |  +--:(tls)
    |     +--rw tls
    |        +--rw endpoints
    |        |  +--rw endpoint* [name]
    |        |     |  +--rw name string
    |        |     +--rw address inet:host
    |        |     +--rw port? inet:port-number
    |        +--rw certificates
    |        |  +--rw certificate* [name]
    |        |     |  +--rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
    |        |     +--rw client-auth
    |        |     +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
    |        |     +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
    |        +--rw cert-maps
    |            +--rw cert-to-name* [id]

Watsen & Schoenwaelder  Expires January 7, 2016  [Page 75]
A.5.2. Example Usage

TBD

A.5.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-restconf-server-new@2015-07-06.yang"

module ietf-restconf-server-new {
  yang-version 1.1;

  prefix "rcserver";

  import ietf-netconf-acm {
    prefix nacm;                             // RFC 6536
  }
  import ietf-inet-types {                   // RFC 6991
    prefix inet;
  }
  import ietf-x509-cert-to-name {            // RFC 7407
    prefix x509c2n;
  }
  import ietf-tls-server {                   // RFC VVVV
    prefix ts;
  }
}

<CODE ENDS>
organization
"IETF NETCONF (Network Configuration) Working Group";

contact
"WG Web: <http://tools.ietf.org/wg/netconf/>
WG List: <mailto:netconf@ietf.org>
WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nsn.com>
WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com>
Editor: Kent Watsen
<mailto:kwatsen@juniper.net>";

description
"This module contains a collection of YANG definitions for configuring RESTCONF servers.

Copyright (c) 2014 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 VVVV; see the RFC itself for full legal notices.";

revision "2015-07-06" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

feature tls-listen {
  description
    "The listen feature indicates that the RESTCONF server...";
supports opening a port to listen for incoming RESTCONF client connections.

feature tls-call-home {
  description
  "The call-home feature indicates that the RESTCONF server supports initiating connections to RESTCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home"
}

feature client-cert-auth {
  description
  "The client-cert-auth feature indicates that the RESTCONF server supports the ClientCertificate authentication scheme.";
  reference
  "RFC ZZZZ: Client Authentication over New TLS Connection"
}

// top-level container (groupings below)
container restconf-server {
  description
  "Top-level container for RESTCONF server configuration."
}

container listen {
  description
  "Configures listen behavior";
  if-feature tls-listen;
  leaf max-sessions {
    type uint16;
    default 0; // should this be 'max'?
    description
    "Specifies the maximum number of concurrent sessions that can be active at one time. The value 0 indicates that no artificial session limit should be used."
  }
}

list endpoint {
  key name;
  description
  "List of endpoints to listen for RESTCONF connections on."
  leaf name {
    type string;
    description
    "An arbitrary name for the RESTCONF listen endpoint."
  }
}
choice transport {
  mandatory true;
  description
    "Selects between available transports."
  case tls {
    container tls {
      description
        "TLS-specific listening configuration for inbound
         connections."
      leaf address {
        type inet:ip-address;
        description
          "The IP address of the interface to listen on. The
           RESTCONF server will listen on all interfaces if
           no value is specified."
      }
      leaf port {
        type inet:port-number;
        default 443;
        description
          "The port number the RESTCONF server will listen on."
      }
      uses tls-server-grouping;
    }
  }
}

container call-home {
  if-feature tls-call-home;
  description
    "Configures call-home behavior"
  list restconf-client {
    key name;
    description
      "List of RESTCONF clients the RESTCONF server is to
       initiate call-home connections to."
    leaf name {
      type string;
      description
        "An arbitrary name for the remote RESTCONF client."
    }
    choice transport {
      mandatory true;
      description
        "Selects between TLS and any transports augmented in."
    }
  }
}
case tls {
  container tls {
    description
    "Specifies TLS-specific call-home transport configuration.";
    container endpoints {
      description
      "Container for the list of endpoints.";
      list endpoint {
        key name;
        min-elements 1;
        ordered-by user;
        description
        "User-ordered list of endpoints for this RESTCONF client. More than one enables high-availability.";
        leaf name {
          type string;
          description
          "An arbitrary name for this endpoint.";
        }
        leaf address {
          type inet:host;
          mandatory true;
          description
          "The IP address or hostname of the endpoint. If a hostname is configured and the DNS resolution results in more than one IP address, the RESTCONF server will process the IP addresses as if they had been explicitly configured in place of the hostname.";
        }
        leaf port {
          type inet:port-number;
          default 9999;
          description
          "The IP port for this endpoint. The RESTCONF server will use the IANA-assigned well-known port if no value is specified.";
        }
      }
    }
    uses tls-server-grouping;
  }
  container connection-type {
    description
    "Indicates the RESTCONF client’s preference for how the
RESTCONF server’s connection is maintained.

choice connection-type {
    description
    "Selects between available connection types."
    case persistent-connection {
        container persistent {
            presence true;
            description
            "Maintain a persistent connection to the RESTCONF client. If the connection goes down, immediately start trying to reconnect to it, using the reconnection strategy.

            This connection type minimizes any RESTCONF client to RESTCONF server data-transfer delay, albeit at the expense of holding resources longer."
        }
        container keep-alives {
            description
            "Configures the keep-alive policy, to proactively test the aliveness of the TLS client. An unresponsive TLS client will be dropped after approximately (max-attempts * max-wait) seconds."
            reference
            "RFC YYYY: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6"
            leaf max-wait {
                type uint16 {
                    range "1..max"
                }
                units seconds;
                default 30;
                description
                "Sets the amount of time in seconds after which if no data has been received from the TLS client, a TLS-level message will be sent to test the aliveness of the TLS client."
            }
            leaf max-attempts {
                type uint8;
                default 3;
                description
                "Sets the number of sequential keep-alive messages that can fail to obtain a response from the TLS client before assuming the TLS client is no longer alive."
            }
        }
    }
}
case periodic-connection {
  container periodic {
    presence true;
    description
    "Periodically connect to the RESTCONF client, so that
    the RESTCONF client may deliver messages pending for
    the RESTCONF server. The RESTCONF client is expected
to close the connection when it is ready to release
it, thus starting the RESTCONF server’s timer until
next connection.";
    leaf reconnect-timeout {
      type uint16 {
        range "1..max";
      }
      units minutes;
      default 60;
      description
      "The maximum amount of unconnected time the RESTCONF
server will wait before re-establishing a connection
 to the RESTCONF client. The RESTCONF server may
initiate a connection before this time if desired
(e.g., to deliver a notification).";
    }
  }
}
}
}
}
}
}
container reconnect-strategy {
  description
  "The reconnection strategy guides how a RESTCONF server
reconnects to an RESTCONF client, after losing a connection
to it, even if due to a reboot. The RESTCONF server starts
with the specified endpoint and tries to connect to it
max-attempts times before trying the next endpoint in the
list (round robin).";
  leaf start-with {
    type enumeration {
      enum first-listed {
        description
        "Indicates that reconnections should start with
the first endpoint listed.";
      }
      enum last-connected {
        description
        "Indicates that reconnections should start with
the endpoint last connected to. If no previous
connection has ever been established, then the
first endpoint configured is used. RESTCONF
servers SHOULD be able to remember the last
endpoint connected to across reboots.

leaf max-attempts {
  type uint8 {
    range "1..max";
  }
  default 3;
  description
  "Specifies the number times the RESTCONF server tries to
  connect to a specific endpoint before moving on to the
  next endpoint in the list (round robin).";
}

grouping tls-server-grouping {
  description
  "An augmentation of tls-server-grouping, as defined in the
  ietf-tls-server module, to add in cert-maps.";
  uses ts:tls-server-grouping {
    augment "client-auth" {
      container cert-maps {
        uses x509c2n:cert-to-name;
        description
        "The cert-maps container is used by a NETCONF server to
        map the NETCONF client’s presented X.509 certificate to a
        NETCONF username. If no matching and valid cert-to-name
        list entry can be found, then the NETCONF server MUST
        close the connection, and MUST NOT accept NETCONF
        messages over it.";
        reference
        "RFC WWWW: NETCONF over TLS, Section 7";
      }
    }
  }
}
Appendix B. Change Log

B.1. 00 to 01

- Restructured document so it flows better
- Added trusted-ca-certs and trusted-client-certs objects into the
  ietf-system-tls-auth module

B.2. 01 to 02

- removed the "one-to-many" construct
- removed "address" as a key field
- removed "network-manager" terminology
- moved open issues to github issues
- brought TLS client auth back into model

B.3. 02 to 03

- fixed tree diagrams and surrounding text

B.4. 03 to 04

- reduced the number of grouping statements
- removed psk-maps and associated feature statements
- added ability for listen/call-home instances to specify which
  host-keys/certificates (of all listed) to use
- clarified that last-connected should span reboots
- added missing "objectives" for selecting which keys to use,
  authenticating client-certificates, and mapping authenticated
  client-certificates to usernames
- clarified indirect client certificate authentication
- added keep-alive configuration for listen connections
o added global-level NETCONF session parameters

B.5. 04 to 05

o Removed all refs to the old ietf-system-tls-auth module

o Removed YANG 1.1 style if-feature statements (loss some expressiveness)

o Removed the read-only (config false) lists of SSH host-keys and TLS certs

o Added an if-feature around session-options container

o Added ability to configure trust-anchors for SSH X.509 client certs

o Now imports by revision, per best practice

o Added support for RESTCONF server

o Added RFC Editor instructions

B.6. 05 to 06

o Removed feature statement on the session-options container (issue #21).

o Added NACM statements to YANG modules for sensitive nodes (issue #24).

o Fixed default RESTCONF server port value to be 443 (issue #26).

o Added client-cert-auth subtree to ietf-restconf-server module (issue #27).

o Updated draft-ietf-netmod-snmp-cfg reference to RFC 7407 (issue #28).

o Added description statements for groupings (issue #29).

o Added description for braces to tree diagram section (issue #30).

o Renamed feature from "rfc6187" to "ssh-x509-certs" (issue #31).
B.7. 06 to 07

- Replaced "application" with "NETCONF/RESTCONF client" (issue #32).
- Reverted back to YANG 1.1 if-feature statements (issue #34).
- Removed import by revisions (issue #36).
- Removed groupings only used once (issue #37).
- Removed upper-bound on hello-timeout, idle-timeout, and max-sessions (issue #38).
- Clarified that when no listen address is configured, the NETCONF/RESTCONF server will listen on all addresses (issue #41).
- Update keep-alive reference to new section in Call Home draft (issue #42).
- Modified connection-type/persistent/keep-alives/interval-secs default value, removed the connection-type/periodic/linger-secs node, and also removed the reconnect-strategy/interval-secs node (issue #43).
- Clarified how last-connected reconnection type should work across reboots (issue #44).
- Clarified how DNS-expanded hostnames should be processed (issue #45).
- Removed text on how to implement keep-alives (now in the call-home draft) and removed the keep-alive configuration for listen connections (issue #46).
- Clarified text for .../periodic-connection/timeout-mins (issue #47).
- Fixed description on the "trusted-ca-certs" leaf-list (issue #48).
- Added optional keychain-based solution in appendix A (issue #49).
- Fixed description text for the interval-secs leaf (issue #50).
- Moved idle-time into the listen, persistent, and periodic subtrees (issue #51).
- Put presence statements on containers where it makes sense (issue #53).
Appendix C. Open Issues

Please see: https://github.com/netconf-wg/server-model/issues.

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de