

Internet Engineering Task Force
Internet-Draft
Obsoletes: 6536 (if approved)
Intended status: Standards Track
Expires: April 29, 2017

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
October 26, 2016

Network Configuration Protocol (NETCONF) Access Control Model
draft-bierman-netconf-rfc6536bis-00

Abstract

The standardization of network configuration interfaces for use with the Network Configuration Protocol (NETCONF) or RESTCONF protocol requires a structured and secure operating environment that promotes human usability and multi-vendor interoperability. There is a need for standard mechanisms to restrict NETCONF or RESTCONF protocol access for particular users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content. This document defines such an access control model.

This document obsoletes RFC 6536.

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 April 29, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Changes Since RFC 6535	5
2. Access Control Design Objectives	5
2.1. Access Control Points	6
2.2. Simplicity	6
2.3. Procedural Interface	7
2.4. Datastore Access	7
2.5. Users and Groups	7
2.6. Maintenance	7
2.7. Configuration Capabilities	8
2.8. Identifying Security-Sensitive Content	8
3. NETCONF Access Control Model (NACM)	9
3.1. Introduction	9
3.1.1. Features	9
3.1.2. External Dependencies	10
3.1.3. Message Processing Model	10
3.2. Datastore Access	13
3.2.1. Access Rights	13
3.2.2. RESTCONF Methods	13
3.2.3. <get> and <get-config> Operations	14
3.2.4. <edit-config> Operation	14
3.2.5. <copy-config> Operation	15
3.2.6. <delete-config> Operation	16
3.2.7. <commit> Operation	16
3.2.8. <discard-changes> Operation	16
3.2.9. <kill-session> Operation	17
3.3. Model Components	17
3.3.1. Users	17
3.3.2. Groups	17
3.3.3. Emergency Recovery Session	17
3.3.4. Global Enforcement Controls	18
3.3.4.1. enable-nacm Switch	18
3.3.4.2. read-default Switch	18
3.3.4.3. write-default Switch	18
3.3.4.4. exec-default Switch	19
3.3.4.5. enable-external-groups Switch	19
3.3.5. Access Control Rules	19
3.4. Access Control Enforcement Procedures	19

3.4.1.	Initial Operation	20
3.4.2.	Session Establishment	20
3.4.3.	"access-denied" Error Handling	20
3.4.4.	Incoming RPC Message Validation	20
3.4.5.	Data Node Access Validation	23
3.4.6.	Outgoing <notification> Authorization	25
3.5.	Data Model Definitions	27
3.5.1.	Data Organization	28
3.5.2.	YANG Module	28
3.6.	IANA Considerations	38
3.7.	Security Considerations	38
3.7.1.	NACM Configuration and Monitoring Considerations	39
3.7.2.	General Configuration Issues	40
3.7.3.	Data Model Design Considerations	42
4.	References	42
4.1.	Normative References	42
4.2.	Informative References	43
Appendix A.	Usage Examples	44
A.1.	<groups> Example	44
A.2.	Module Rule Example	45
A.3.	Protocol Operation Rule Example	46
A.4.	Data Node Rule Example	48
A.5.	Notification Rule Example	50
Authors' Addresses	51

1. Introduction

The NETCONF and RESTCONF protocols do not provide any standard mechanisms to restrict the protocol operations and content that each user is authorized to access.

There is a need for interoperable management of the controlled access to administrator-selected portions of the available NETCONF or RESTCONF content within a particular server.

This document addresses access control mechanisms for the Operations and Content layers of NETCONF, as defined in [RFC6241], and RESTCONF, as defined in [I-D.ietf-netconf-restconf]. It contains three main sections:

1. Access Control Design Objectives
2. NETCONF Access Control Model (NACM)
3. YANG Data Model (ietf-netconf-acm.yang)

YANG version 1.1 [RFC7950] adds two new constructs that need special access control handling. The "action" statement is similar to the

"rpc" statement, except it is located within a data node. The "notification" statement can also be located within a data node.

1.1. Terminology

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

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

- o client
- o datastore
- o protocol operation
- o server
- o session
- o user

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

- o action
- o data node
- o data definition statement

The following terms are defined in [I-D.ietf-netconf-restconf]. and are not redefined here:

- o data resource
- o datastore resource
- o operation resource

The following terms are used throughout this document:

access control: A security feature provided by the NETCONF server that allows an administrator to restrict access to a subset of all NETCONF protocol operations and data, based on various criteria.

access control model (ACM): A conceptual model used to configure and monitor the access control procedures desired by the administrator to enforce a particular access control policy.

access control rule: The criterion used to determine if a particular NETCONF protocol operation will be permitted or denied.

access operation: How a request attempts to access a conceptual object. One of "none", "read", "create", "delete", "update", or "execute".

data node hierarchy: The hierarchy of data nodes that identifies the specific "action" or "notification" node in the datastore.

recovery session: A special administrative session that is given unlimited NETCONF access and is exempt from all access control enforcement. The mechanism(s) used by a server to control and identify whether or not a session is a recovery session are implementation specific and outside the scope of this document.

write access: A shorthand for the "create", "delete", and "update" access operations.

1.2. Changes Since RFC 6535

The NACM procedures and data model have been updated to support new data modeling capabilities in the version 1.1. of the YANG data modeling language. The "action" and "notification" statements can be used within data nodes to define data-model specific operations and notifications.

An important use-case for these new YANG statements is the increased access control granularity that can be achieved over top-level "rpc" and "notification" statements. The new "action" and "notification" statements are used within data nodes, and access to the action or notification can be restricted to specific instances of these data nodes.

Support for the RESTCONF protocol has been added. The RESTCONF operations are similar to the NETCONF operations, so a simple mapping to the existing NACM procedures and data model is possible.

2. Access Control Design Objectives

This section documents the design objectives for the NETCONF Access Control Model presented in Section 3.

2.1. Access Control Points

NETCONF allows new protocol operations to be added at any time, and the YANG Data Modeling Language supports this feature. It is not possible to design an ACM for NETCONF that only focuses on a static set of protocol operations, like some other protocols. Since few assumptions can be made about an arbitrary protocol operation, the NETCONF architectural server components need to be protected at three conceptual control points.

These access control points, described in Figure 1, are as follows:

protocol operation: Permission to invoke specific protocol operations.

datastore: Permission to read and/or alter specific data nodes within any datastore.

notification: Permission to receive specific notification event types.

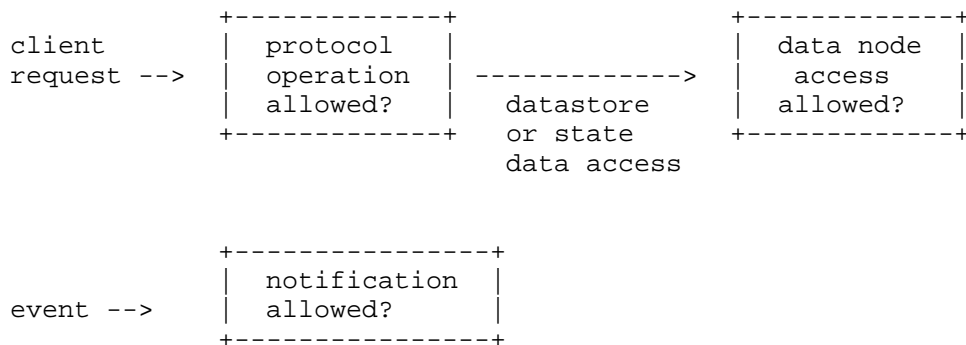


Figure 1

2.2. Simplicity

There is concern that a complicated ACM will not be widely deployed because it is too hard to use. It needs to be easy to do simple things and possible to do complex things, instead of hard to do everything.

Configuration of the access control system needs to be as simple as possible. Simple and common tasks need to be easy to configure and require little expertise or domain-specific knowledge. Complex tasks are possible using additional mechanisms, which may require additional expertise.

A single set of access control rules ought to be able to control all types of NETCONF protocol operation invocation, all datastore access, and all notification events.

Access control ought to be defined with a small and familiar set of permissions, while still allowing full control of NETCONF datastore access.

2.3. Procedural Interface

The NETCONF protocol uses a remote procedure call model and an extensible set of protocol operations. Access control for any possible protocol operation is necessary.

2.4. Datastore Access

It is necessary to control access to specific nodes and subtrees within the NETCONF datastore, regardless of which protocol operation, standard or proprietary, was used to access the datastore.

2.5. Users and Groups

It is necessary that access control rules for a single user or a configurable group of users can be configured.

The ACM needs to support the concept of administrative groups, to support the well-established distinction between a root account and other types of less-privileged conceptual user accounts. These groups need to be configurable by the administrator.

It is necessary that the user-to-group mapping can be delegated to a central server, such as a RADIUS server [RFC2865][RFC5607]. Since authentication is performed by the NETCONF transport layer and RADIUS performs authentication and service authorization at the same time, the underlying NETCONF transport needs to be able to report a set of group names associated with the user to the server. It is necessary that the administrator can disable the usage of these group names within the ACM.

2.6. Maintenance

It ought to be possible to disable part or all of the access control model enforcement procedures without deleting any access control rules.

2.7. Configuration Capabilities

Suitable configuration and monitoring mechanisms are needed to allow an administrator to easily manage all aspects of the ACM's behavior. A standard data model, suitable for use with the <edit-config> protocol operation, needs to be available for this purpose.

Access control rules to restrict access operations on specific subtrees within the configuration datastore need to be supported.

2.8. Identifying Security-Sensitive Content

One of the most important aspects of the data model documentation, and biggest concerns during deployment, is the identification of security-sensitive content. This applies to protocol operations in NETCONF, not just data and notifications.

It is mandatory for security-sensitive objects to be documented in the Security Considerations section of an RFC. This is nice, but it is not good enough, for the following reasons:

- o This documentation-only approach forces administrators to study the RFC and determine if there are any potential security risks introduced by a new data model.
- o If any security risks are identified, then the administrator must study some more RFC text and determine how to mitigate the security risk(s).
- o The ACM on each server must be configured to mitigate the security risks, e.g., require privileged access to read or write the specific data identified in the Security Considerations section.
- o If the ACM is not pre-configured, then there will be a time window of vulnerability after the new data model is loaded and before the new access control rules for that data model are configured, enabled, and debugged.

Often, the administrator just wants to disable default access to the secure content, so no inadvertent or malicious changes can be made to the server. This allows the default rules to be more lenient, without significantly increasing the security risk.

A data model designer needs to be able to use machine-readable statements to identify NETCONF content, which needs to be protected by default. This will allow client and server tools to automatically identify data-model-specific security risks, by denying access to

sensitive data unless the user is explicitly authorized to perform the requested access operation.

3. NETCONF Access Control Model (NACM)

3.1. Introduction

This section provides a high-level overview of the access control model structure. It describes the NETCONF protocol message processing model and the conceptual access control requirements within that model.

3.1.1. Features

The NACM data model provides the following features:

- o Independent control of remote procedure call (RPC), action, data, and notification access.
- o Simple access control rules configuration data model that is easy to use.
- o The concept of an emergency recovery session is supported, but configuration of the server for this purpose is beyond the scope of this document. An emergency recovery session will bypass all access control enforcement, in order to allow it to initialize or repair the NACM configuration.
- o A simple and familiar set of datastore permissions is used.
- o Support for YANG security tagging (e.g., "nacm:default-deny-write" statement) allows default security modes to automatically exclude sensitive data.
- o Separate default access modes for read, write, and execute permissions.
- o Access control rules are applied to configurable groups of users.
- o The access control enforcement procedures can be disabled during operation, without deleting any access control rules, in order to debug operational problems.
- o Access control rules are simple to configure.
- o The number of denied protocol operation requests and denied datastore write requests can be monitored by the client.

- o Simple unconstrained YANG instance identifiers are used to configure access control rules for specific data nodes.

3.1.2. External Dependencies

The NETCONF protocol [RFC6241] is used for network management purposes within this document.

The RESTCONF protocol [I-D.ietf-netconf-restconf] is used for network management purposes within this document.

The YANG Data Modeling Language [RFC7950] is used to define the data models for use with the NETCONF or RESTCONF protocols. YANG is also used to define the data model in this document.

3.1.3. Message Processing Model

The following diagram shows the conceptual message flow model, including the points at which access control is applied during NETCONF message processing.

RESTCONF operations are mapped to the access control model based on the HTTP method and resource class used in the operation. For example, a POST method on a data resource is considered "write data node" access, but a POST method on an operation resource is considered "operation" access.

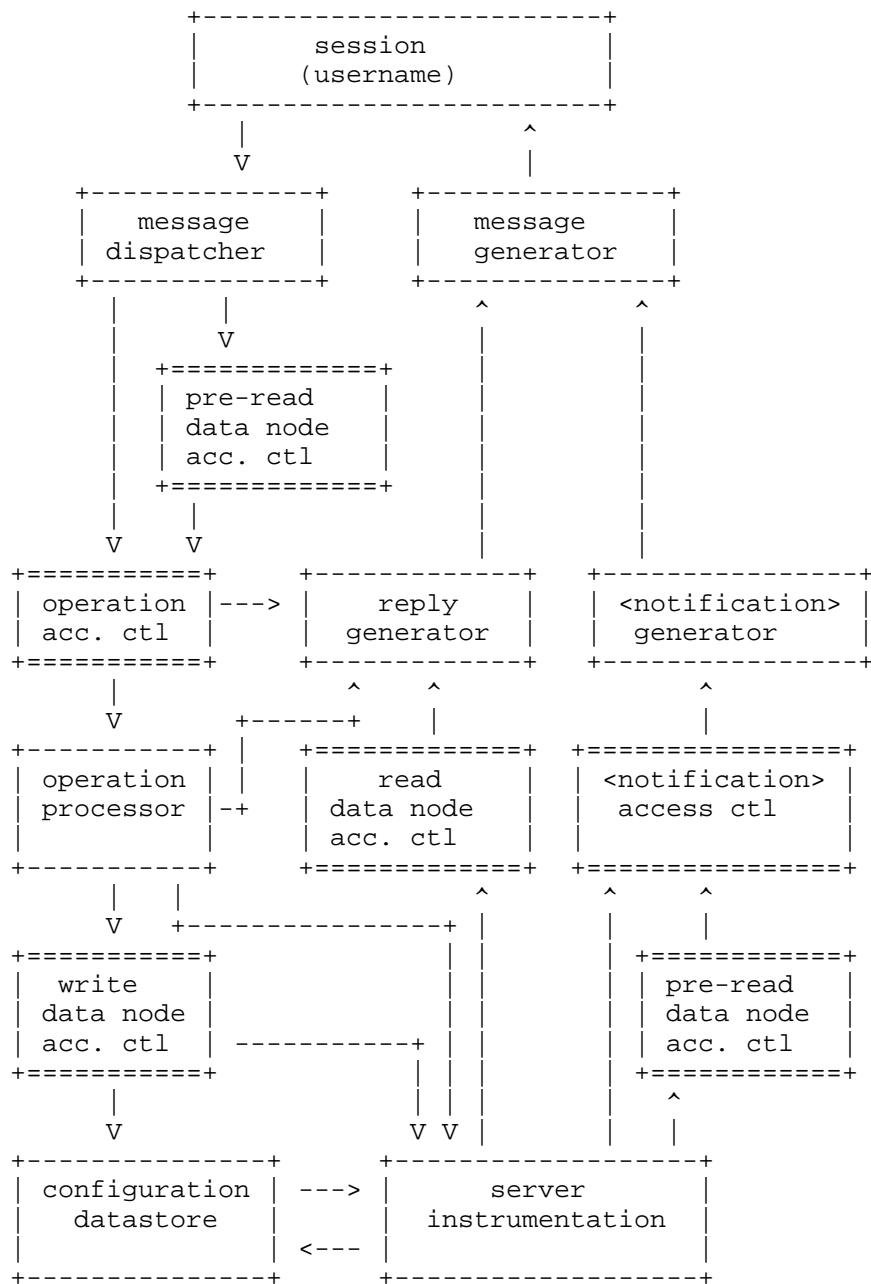


Figure 2

The following high-level sequence of conceptual processing steps is executed for each received <rpc> message, if access control enforcement is enabled:

- o For each active session, access control is applied individually to all <rpc> messages (except <close-session>) received by the server, unless the session is identified as a recovery session.
- o If the <action> operation defined in [RFC7950] is invoked, then read access is required for all instances in the hierarchy of data nodes that identifies the specific action in the datastore, and execute access is required for the action node. If the user is not authorized to read all the specified data nodes and execute the action, then the request is rejected with an "access-denied" error.
- o Otherwise, if the user is not authorized to execute the specified protocol operation, then the request is rejected with an "access-denied" error.
- o If the configuration datastore or conceptual state data is accessed by the protocol operation, then the server checks if the client is authorized to access the nodes in the datastore. If the user is not authorized to perform the requested access operation on the requested data, then the request is rejected with an "access-denied" error.

The following sequence of conceptual processing steps is executed for each generated notification event, if access control enforcement is enabled:

- o Server instrumentation generates a notification for a particular subscription.
- o If the notification statement is specified within a data subtree, as specified in [RFC7950], then read access is required for all instances in the hierarchy of data nodes that identifies the specific notification in the datastore, and read access is required for the notification node. If the user is not authorized to read all the specified data nodes and the notification node, then the notification is dropped for that subscription.
- o If the notification statement is a top-level statement, the notification access control enforcer checks the notification event type, and if it is one that the user is not authorized to read, then the notification is dropped for that subscription.

3.2. Datastore Access

The same access control rules apply to all datastores, for example, the candidate configuration datastore or the running configuration datastore.

Only the standard NETCONF datastores (candidate, running, and startup) are controlled by NACM. Local or remote files or datastores accessed via the <url> parameter are not controlled by NACM. A standalone RESTCONF server (i.e., not co-located with a NETCONF server) applies NACM rules to a conceptual datastore, since datastores are not supported in RESTCONF.

3.2.1. Access Rights

A small set of hard-wired datastore access rights is needed to control access to all possible NETCONF protocol operations, including vendor extensions to the standard protocol operation set.

The "CRUDX" model can support all NETCONF protocol operations:

- o Create: allows the client to add a new data node instance to a datastore.
- o Read: allows the client to read a data node instance from a datastore or receive the notification event type.
- o Update: allows the client to update an existing data node instance in a datastore.
- o Delete: allows the client to delete a data node instance from a datastore.
- o eXec: allows the client to execute the operation.

3.2.2. RESTCONF Methods

The RESTCONF protocol utilizes HTTP methods to perform datastore operations, similar to the NETCONF protocol. The NACM procedures were originally written for NETCONF protocol operations so the RESTCONF methods are mapped to NETCONF operations for the purpose of access control processing. The enforcement procedures described within this document apply to both protocols unless explicitly stated otherwise.

Not all RESTCONF methods are subject to access control. The following table specifies how each method is mapped to NETCONF

protocol operations. The value 'none' indicates that NACM is not applied at all to the specific RESTCONF method.

method	resource class	NETCONF operation	Edit operation
OPTIONS	all	none	N/A
HEAD	all	<get>	N/A
GET	all	<get>	N/A
POST	datastore, data	<edit-config>	create
POST	operation	specified operation	N/A
PUT	data	<edit-config>	create, replace
PUT	datastore	<copy-config>	replace
PATCH	data, datastore	<edit-config>	merge
DELETE	data	<edit-config>	delete

Table 1: Mapping RESTCONF Methods to NETCONF

3.2.3. <get> and <get-config> Operations

Data nodes to which the client does not have read access are silently omitted from the <rpc-reply> message. This is done to allow NETCONF filters for <get> and <get-config> to function properly, instead of causing an "access-denied" error because the filter criteria would otherwise include unauthorized read access to some data nodes. For NETCONF filtering purposes, the selection criteria is applied to the subset of nodes that the user is authorized to read, not the entire datastore.

3.2.4. <edit-config> Operation

The NACM access rights are not directly coupled to the <edit-config> "operation" attribute, although they are similar. Instead, a NACM access right applies to all protocol operations that would result in a particular access operation to the target datastore. This section describes how these access rights apply to the specific access operations supported by the <edit-config> protocol operation.

If the effective access operation is "none" (i.e., default-operation="none") for a particular data node, then no access control is applied to that data node. This is required to allow access to a subtree within a larger data structure. For example, a user may be authorized to create a new "/interfaces/interface" list entry but not be authorized to create or delete its parent container ("/interfaces"). If the "/interfaces" container already exists in the target datastore, then the effective operation will be "none" for

the `"/interfaces"` node if an `"/interfaces/interface"` list entry is edited.

If the protocol operation would result in the creation of a datastore node and the user does not have "create" access permission for that node, the protocol operation is rejected with an "access-denied" error.

If the protocol operation would result in the deletion of a datastore node and the user does not have "delete" access permission for that node, the protocol operation is rejected with an "access-denied" error.

If the protocol operation would result in the modification of a datastore node and the user does not have "update" access permission for that node, the protocol operation is rejected with an "access-denied" error.

A "merge" or "replace" `<edit-config>` operation may include data nodes that do not alter portions of the existing datastore. For example, a container or list node may be present for naming purposes but does not actually alter the corresponding datastore node. These unaltered data nodes are ignored by the server and do not require any access rights by the client.

A "merge" `<edit-config>` operation may include data nodes but not include particular child data nodes that are present in the datastore. These missing data nodes within the scope of a "merge" `<edit-config>` operation are ignored by the server and do not require any access rights by the client.

The contents of specific restricted datastore nodes MUST NOT be exposed in any `<rpc-error>` elements within the reply.

3.2.5. `<copy-config>` Operation

Access control for the `<copy-config>` protocol operation requires special consideration because the administrator may be replacing the entire target datastore.

If the source of the `<copy-config>` protocol operation is the running configuration datastore and the target is the startup configuration datastore, the client is only required to have permission to execute the `<copy-config>` protocol operation.

Otherwise:

- o If the source of the <copy-config> operation is a datastore, then data nodes to which the client does not have read access are silently omitted.
- o If the target of the <copy-config> operation is a datastore, the client needs access to the modified nodes, specifically:
 - * If the protocol operation would result in the creation of a datastore node and the user does not have "create" access permission for that node, the protocol operation is rejected with an "access-denied" error.
 - * If the protocol operation would result in the deletion of a datastore node and the user does not have "delete" access permission for that node, the protocol operation is rejected with an "access-denied" error.
 - * If the protocol operation would result in the modification of a datastore node and the user does not have "update" access permission for that node, the protocol operation is rejected with an "access-denied" error.

3.2.6. <delete-config> Operation

Access to the <delete-config> protocol operation is denied by default. The "exec-default" leaf does not apply to this protocol operation. Access control rules must be explicitly configured to allow invocation by a non-recovery session.

3.2.7. <commit> Operation

The server MUST determine the exact nodes in the running configuration datastore that are actually different and only check "create", "update", and "delete" access permissions for this set of nodes, which could be empty.

For example, if a session can read the entire datastore but only change one leaf, that session needs to be able to edit and commit that one leaf.

3.2.8. <discard-changes> Operation

The client is only required to have permission to execute the <discard-changes> protocol operation. No datastore permissions are needed.

3.2.9. <kill-session> Operation

The <kill-session> operation does not directly alter a datastore. However, it allows one session to disrupt another session that is editing a datastore.

Access to the <kill-session> protocol operation is denied by default. The "exec-default" leaf does not apply to this protocol operation. Access control rules must be explicitly configured to allow invocation by a non-recovery session.

3.3. Model Components

This section defines the conceptual components related to the access control model.

3.3.1. Users

A "user" is the conceptual entity that is associated with the access permissions granted to a particular session. A user is identified by a string that is unique within the server.

As described in [RFC6241], the username string is derived from the transport layer during session establishment. If the transport layer cannot authenticate the user, the session is terminated.

3.3.2. Groups

Access to a specific NETCONF protocol operation is granted to a session, associated with a group, not a user.

A group is identified by its name. All group names are unique within the server.

A group member is identified by a username string.

The same user can be a member of multiple groups.

3.3.3. Emergency Recovery Session

The server MAY support a recovery session mechanism, which will bypass all access control enforcement. This is useful for restricting initial access and repairing a broken access control configuration.

3.3.4. Global Enforcement Controls

There are five global controls that are used to help control how access control is enforced.

3.3.4.1. enable-nacm Switch

A global "enable-nacm" on/off switch is provided to enable or disable all access control enforcement. When this global switch is set to "true", then all requests are checked against the access control rules and only permitted if configured to allow the specific access request. When this global switch is set to "false", then all access requested are permitted.

3.3.4.2. read-default Switch

An on/off "read-default" switch is provided to enable or disable default access to receive data in replies and notifications. When the "enable-nacm" global switch is set to "true", then this global switch is relevant if no matching access control rule is found to explicitly permit or deny read access to the requested NETCONF datastore data or notification event type.

When this global switch is set to "permit" and no matching access control rule is found for the NETCONF datastore read or notification event requested, then access is permitted.

When this global switch is set to "deny" and no matching access control rule is found for the NETCONF datastore read or notification event requested, then access is denied.

3.3.4.3. write-default Switch

An on/off "write-default" switch is provided to enable or disable default access to alter configuration data. When the "enable-nacm" global switch is set to "true", then this global switch is relevant if no matching access control rule is found to explicitly permit or deny write access to the requested NETCONF datastore data.

When this global switch is set to "permit" and no matching access control rule is found for the NETCONF datastore write requested, then access is permitted.

When this global switch is set to "deny" and no matching access control rule is found for the NETCONF datastore write requested, then access is denied.

3.3.4.4. exec-default Switch

An on/off "exec-default" switch is provided to enable or disable default access to execute protocol operations. When the "enable-nacm" global switch is set to "true", then this global switch is relevant if no matching access control rule is found to explicitly permit or deny access to the requested NETCONF protocol operation.

When this global switch is set to "permit" and no matching access control rule is found for the NETCONF protocol operation requested, then access is permitted.

When this global switch is set to "deny" and no matching access control rule is found for the NETCONF protocol operation requested, then access is denied.

3.3.4.5. enable-external-groups Switch

When this global switch is set to "true", the group names reported by the NETCONF transport layer for a session are used together with the locally configured group names to determine the access control rules for the session.

When this switch is set to "false", the group names reported by the NETCONF transport layer are ignored by NACM.

3.3.5. Access Control Rules

There are four types of rules available in NACM:

module rule: controls access for definitions in a specific YANG module, identified by its name.

protocol operation rule: controls access for a specific protocol operation, identified by its YANG module and name.

data node rule: controls access for a specific data node, identified by its path location within the conceptual XML document for the data node.

notification rule: controls access for a specific notification event type, identified by its YANG module and name.

3.4. Access Control Enforcement Procedures

There are seven separate phases that need to be addressed, four of which are related to the NETCONF message processing model (Section 3.1.3). In addition, the initial startup mode for a NETCONF

server, session establishment, and "access-denied" error-handling procedures also need to be considered.

The server MUST use the access control rules in effect at the time it starts processing the message. The same access control rules MUST stay in effect for the processing of the entire message.

3.4.1. Initial Operation

Upon the very first startup of the NETCONF server, the access control configuration will probably not be present. If it isn't, a server MUST NOT allow any write access to any session role except a recovery session.

Access rules are enforced any time a request is initiated from a user session. Access control is not enforced for server-initiated access requests, such as the initial load of the running datastore, during bootup.

3.4.2. Session Establishment

The access control model applies specifically to the well-formed XML content transferred between a client and a server after session establishment has been completed and after the <hello> exchange has been successfully completed.

Once session establishment is completed and a user has been authenticated, the NETCONF transport layer reports the username and a possibly empty set of group names associated with the user to the NETCONF server. The NETCONF server will enforce the access control rules, based on the supplied username, group names, and the configuration data stored on the server.

3.4.3. "access-denied" Error Handling

The "access-denied" error-tag is generated when the access control system denies access to either a request to invoke a protocol operation or a request to perform a particular access operation on the configuration datastore.

A server MUST NOT include any information the client is not allowed to read in any <error-info> elements within the <rpc-error> response.

3.4.4. Incoming RPC Message Validation

The diagram below shows the basic conceptual structure of the access control processing model for incoming NETCONF <rpc> messages within a server.

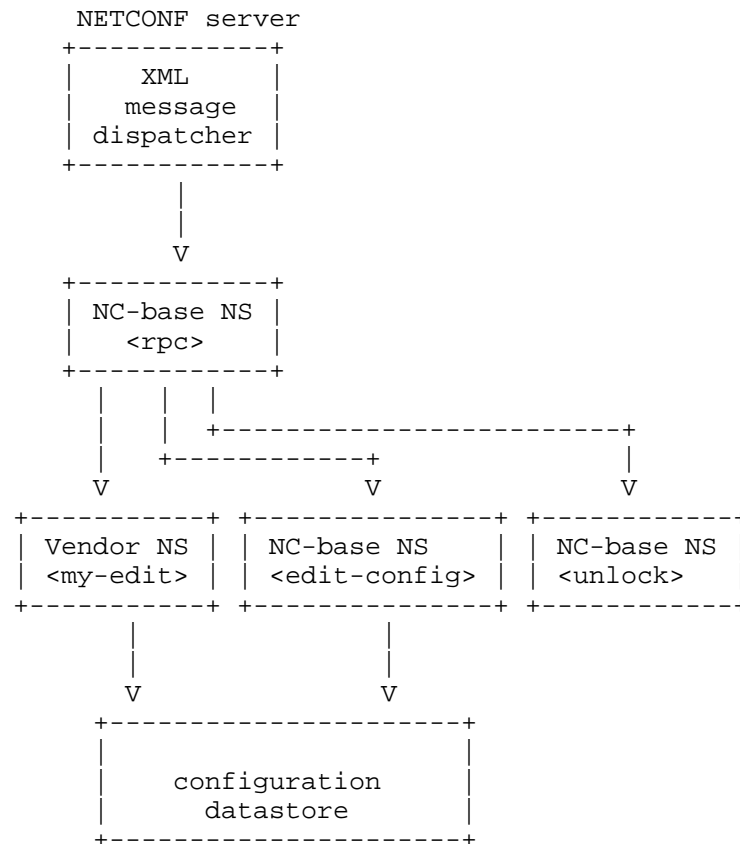


Figure 3

Access control begins with the message dispatcher.

After the server validates the `<rpc>` element and determines the namespace URI and the element name of the protocol operation being requested, the server verifies that the user is authorized to invoke the protocol operation.

The server **MUST** separately authorize every protocol operation by following these steps:

1. If the "enable-nacm" leaf is set to "false", then the protocol operation is permitted.
2. If the requesting session is identified as a recovery session, then the protocol operation is permitted.

3. If the requested operation is the NETCONF <close-session> protocol operation, then the protocol operation is permitted.
4. Check all the "group" entries for ones that contain a "user-name" entry that equals the username for the session making the request. If the "enable-external-groups" leaf is "true", add to these groups the set of groups provided by the transport layer.
5. If no groups are found, continue with step 10.
6. Process all rule-list entries, in the order they appear in the configuration. If a rule-list's "group" leaf-list does not match any of the user's groups, proceed to the next rule-list entry.
7. For each rule-list entry found, process all rules, in order, until a rule that matches the requested access operation is found. A rule matches if all of the following criteria are met:
 - * The rule's "module-name" leaf is "*" or equals the name of the YANG module where the protocol operation is defined.
 - * The rule does not have a "rule-type" defined or the "rule-type" is "protocol-operation" and the "rpc-name" is "*" or equals the name of the requested protocol operation.
 - * The rule's "access-operations" leaf has the "exec" bit set or has the special value "*".
8. If a matching rule is found, then the "action" leaf is checked. If it is equal to "permit", then the protocol operation is permitted; otherwise, it is denied.
9. At this point, no matching rule was found in any rule-list entry.
10. If the requested protocol operation is defined in a YANG module advertised in the server capabilities and the "rpc" statement contains a "nacm:default-deny-all" statement, then the protocol operation is denied.
11. If the requested protocol operation is the NETCONF <kill-session> or <delete-config>, then the protocol operation is denied.
12. If the "exec-default" leaf is set to "permit", then permit the protocol operation; otherwise, deny the request.

If the user is not authorized to invoke the protocol operation, then an `<rpc-error>` is generated with the following information:

error-tag: access-denied

error-path: Identifies the requested protocol operation. The following example represents the `<edit-config>` protocol operation in the NETCONF base namespace:

```
<error-path
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  /nc:rpc/nc:edit-config
</error-path>
```

If a datastore is accessed, either directly or as a side effect of the protocol operation, then the server MUST intercept the access operation and make sure the user is authorized to perform the requested access operation on the specified data, as defined in Section 3.4.5.

3.4.5. Data Node Access Validation

If a data node within a datastore is accessed, or an action or notification tied to a data node, then the server MUST ensure that the user is authorized to perform the requested "read", "create", "update", "delete", or "execute" access operation on the specified data node.

If an action is requested to be executed, the server MUST ensure that the user is authorized to perform the "execute" access operation on the requested action.

If a notification tied to a data node is generated, the server MUST ensure that the user is authorized to perform the "read" access operation on the requested notification.

The data node access request is authorized by following these steps:

1. If the "enable-nacm" leaf is set to "false", then the access operation is permitted.
2. If the requesting session is identified as a recovery session, then the access operation is permitted.
3. Check all the "group" entries for ones that contain a "username" entry that equals the username for the session making the request. If the "enable-external-groups" leaf is "true", add to these groups the set of groups provided by the transport layer.

4. If no groups are found, continue with step 9.
5. Process all rule-list entries, in the order they appear in the configuration. If a rule-list's "group" leaf-list does not match any of the user's groups, proceed to the next rule-list entry.
6. For each rule-list entry found, process all rules, in order, until a rule that matches the requested access operation is found. A rule matches if all of the following criteria are met:
 - * The rule's "module-name" leaf is "*" or equals the name of the YANG module where the requested data node is defined.
 - * The rule does not have a "rule-type" defined or the "rule-type" is "data-node" and the "path" matches the requested data node, action node, or notification node.
 - * For a "read" access operation, the rule's "access-operations" leaf has the "read" bit set or has the special value "*".
 - * For a "create" access operation, the rule's "access-operations" leaf has the "create" bit set or has the special value "*".
 - * For a "delete" access operation, the rule's "access-operations" leaf has the "delete" bit set or has the special value "*".
 - * For an "update" access operation, the rule's "access-operations" leaf has the "update" bit set or has the special value "*".
 - * For an "execute" access operation, the rule's "access-operations" leaf has the "exec" bit set or has the special value "*".
7. If a matching rule is found, then the "action" leaf is checked. If it is equal to "permit", then the data node access is permitted; otherwise, it is denied. For a "read" access operation, "denied" means that the requested data is not returned in the reply.
8. At this point, no matching rule was found in any rule-list entry.
9. For a "read" access operation, if the requested data node is defined in a YANG module advertised in the server capabilities

and the data definition statement contains a "nacm:default-deny-all" statement, then the requested data node is not included in the reply.

10. For a "write" access operation, if the requested data node is defined in a YANG module advertised in the server capabilities and the data definition statement contains a "nacm:default-deny-write" or a "nacm:default-deny-all" statement, then the data node access request is denied.
11. For a "read" access operation, if the "read-default" leaf is set to "permit", then include the requested data node in the reply; otherwise, do not include the requested data node in the reply.
12. For a "write" access operation, if the "write-default" leaf is set to "permit", then permit the data node access request; otherwise, deny the request.
13. For an "execute" access operation, if the "exec-default" leaf is set to "permit", then permit the request; otherwise, deny the request.

3.4.6. Outgoing <notification> Authorization

Configuration of access control rules specifically for descendant nodes of the notification event type element are outside the scope of this document. If the user is authorized to receive the notification event type, then it is also authorized to receive any data it contains.

If the notification is specified within a data subtree, as specified in [RFC7950], then read access to the notification is required. Processing continues as described in Section 3.4.5.

The following figure shows the conceptual message processing model for outgoing <notification> messages.

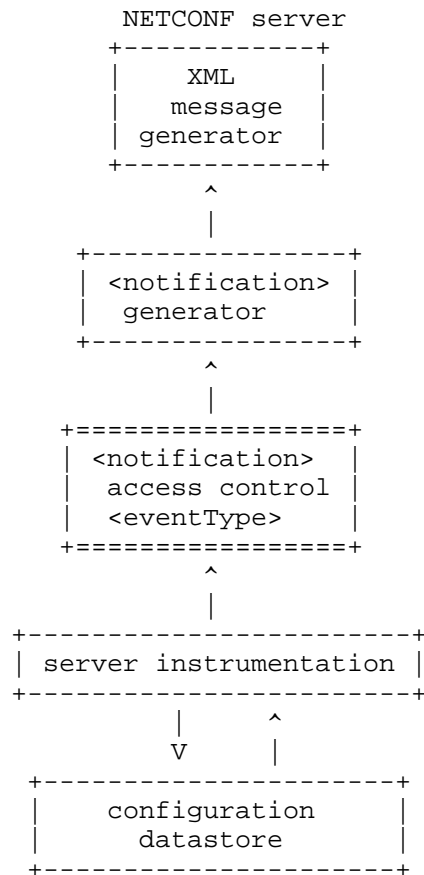


Figure 4

The generation of a notification for a specific subscription [RFC5277] is authorized by following these steps:

1. If the "enable-nacm" leaf is set to "false", then the notification is permitted.
2. If the session is identified as a recovery session, then the notification is permitted.
3. If the notification is the NETCONF <replayComplete> or <notificationComplete> event type [RFC5277], then the notification is permitted.

4. Check all the "group" entries for ones that contain a "user-name" entry that equals the username for the session making the request. If the "enable-external-groups" leaf is "true", add to these groups the set of groups provided by the transport layer.
5. If no groups are found, continue with step 10.
6. Process all rule-list entries, in the order they appear in the configuration. If a rule-list's "group" leaf-list does not match any of the user's groups, proceed to the next rule-list entry.
7. For each rule-list entry found, process all rules, in order, until a rule that matches the requested access operation is found. A rule matches if all of the following criteria are met:
 - * The rule's "module-name" leaf is "*" or equals the name of the YANG module where the notification is defined.
 - * The rule does not have a "rule-type" defined or the "rule-type" is "notification" and the "notification-name" is "*" or equals the name of the notification.
 - * The rule's "access-operations" leaf has the "read" bit set or has the special value "*".
8. If a matching rule is found, then the "action" leaf is checked. If it is equal to "permit", then permit the notification; otherwise, drop the notification for the associated subscription.
9. Otherwise, no matching rule was found in any rule-list entry.
10. If the requested notification is defined in a YANG module advertised in the server capabilities and the "notification" statement contains a "nacm:default-deny-all" statement, then the notification is dropped for the associated subscription.
11. If the "read-default" leaf is set to "permit", then permit the notification; otherwise, drop the notification for the associated subscription.

3.5. Data Model Definitions

3.5.1. Data Organization

The following diagram highlights the contents and structure of the NACM YANG module.

```

module: ietf-netconf-acm
  +--rw nacm
    +--rw enable-nacm?          boolean
    +--rw read-default?         action-type
    +--rw write-default?        action-type
    +--rw exec-default?         action-type
    +--rw enable-external-groups? boolean
    +--ro denied-operations      yang:zero-based-counter32
    +--ro denied-data-writes     yang:zero-based-counter32
    +--ro denied-notifications   yang:zero-based-counter32
    +--rw groups
      | +--rw group* [name]
      |   +--rw name          group-name-type
      |   +--rw user-name*    user-name-type
    +--rw rule-list* [name]
      +--rw name              string
      +--rw group*            union
      +--rw rule* [name]
        +--rw name              string
        +--rw module-name?      union
        +--rw (rule-type)?
          | +--:(protocol-operation)
          | | +--rw rpc-name?      union
          | +--:(notification)
          | | +--rw notification-name? union
          | +--:(data-node)
          | +--rw path              node-instance-identifier
        +--rw access-operations? union
        +--rw action              action-type
        +--rw comment?            string
  
```

3.5.2. YANG Module

The following YANG module specifies the normative NETCONF content that MUST be supported by the server.

The "ietf-netconf-acm" YANG module imports typedefs from [RFC6991].

```

<CODE BEGINS> file "ietf-netconf-acm@2016-08-26.yang"
module ietf-netconf-acm {

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-acm";

```

```
prefix "nacm";

import ietf-yang-types {
  prefix yang;
}

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

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

  Author:     Andy Bierman
               <mailto:andy@yumaworks.com>

  Author:     Martin Bjorklund
               <mailto:mbj@tail-f.com>";

description
  "NETCONF Access Control Model.

  Copyright (c) 2012, 2016 IETF Trust and the persons
  identified as authors of the code. All rights reserved.

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

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

revision "2016-08-26" {
  description
    "Second version";
  reference
    "RFC XXXX: Network Configuration Protocol (NETCONF)
      Access Control Model";
}

revision "2012-02-22" {
  description
    "Initial version";
  reference
```

```
"RFC 6536: Network Configuration Protocol (NETCONF)
    Access Control Model";
}

/*
 * Extension statements
 */

extension default-deny-write {
    description
        "Used to indicate that the data model node
        represents a sensitive security system parameter.

        If present, and the NACM module is enabled (i.e.,
        /nacm/enable-nacm object equals 'true'), the NETCONF server
        will only allow the designated 'recovery session' to have
        write access to the node.  An explicit access control rule is
        required for all other users.

        The 'default-deny-write' extension MAY appear within a data
        definition statement.  It is ignored otherwise.";
}

extension default-deny-all {
    description
        "Used to indicate that the data model node
        controls a very sensitive security system parameter.

        If present, and the NACM module is enabled (i.e.,
        /nacm/enable-nacm object equals 'true'), the NETCONF server
        will only allow the designated 'recovery session' to have
        read, write, or execute access to the node.  An explicit
        access control rule is required for all other users.

        The 'default-deny-all' extension MAY appear within a data
        definition statement, 'rpc' statement, or 'notification'
        statement.  It is ignored otherwise.";
}

/*
 * Derived types
 */

typedef user-name-type {
    type string {
        length "1..max";
    }
    description
```

```
    "General Purpose Username string.";
}

typedef matchall-string-type {
    type string {
        pattern '\*';
    }
    description
        "The string containing a single asterisk '*' is used
        to conceptually represent all possible values
        for the particular leaf using this data type.";
}

typedef access-operations-type {
    type bits {
        bit create {
            description
                "Any protocol operation that creates a
                new data node.";
        }
        bit read {
            description
                "Any protocol operation or notification that
                returns the value of a data node.";
        }
        bit update {
            description
                "Any protocol operation that alters an existing
                data node.";
        }
        bit delete {
            description
                "Any protocol operation that removes a data node.";
        }
        bit exec {
            description
                "Execution access to the specified protocol operation.";
        }
    }
    description
        "NETCONF Access Operation.";
}

typedef group-name-type {
    type string {
        length "1..max";
        pattern '^[^*]*.*';
    }
}
```

```
    description
      "Name of administrative group to which
       users can be assigned.";
  }

  typedef action-type {
    type enumeration {
      enum permit {
        description
          "Requested action is permitted.";
      }
      enum deny {
        description
          "Requested action is denied.";
      }
    }
  }
  description
    "Action taken by the server when a particular
     rule matches.";
}

typedef node-instance-identifier {
  type yang:xpath1.0;
  description
    "Path expression used to represent a special
     data node, action, or notification instance identifier
     string.

    A node-instance-identifier value is an
    unrestricted YANG instance-identifier expression.
    All the same rules as an instance-identifier apply
    except predicates for keys are optional.  If a key
    predicate is missing, then the node-instance-identifier
    represents all possible server instances for that key.

    This XPath expression is evaluated in the following context:

    o The set of namespace declarations are those in scope on
      the leaf element where this type is used.

    o The set of variable bindings contains one variable,
      'USER', which contains the name of the user of the current
      session.

    o The function library is the core function library, but
      note that due to the syntax restrictions of an
      instance-identifier, no functions are allowed.
```



```
o The context node is the root node in the data tree.

The accessible tree includes actions and notifications tied to
data nodes.";
}

/*
 * Data definition statements
 */

container nacm {
    nacm:default-deny-all;

    description
        "Parameters for NETCONF Access Control Model.";

    leaf enable-nacm {
        type boolean;
        default true;
        description
            "Enables or disables all NETCONF access control
            enforcement. If 'true', then enforcement
            is enabled. If 'false', then enforcement
            is disabled.";
    }

    leaf read-default {
        type action-type;
        default "permit";
        description
            "Controls whether read access is granted if
            no appropriate rule is found for a
            particular read request.";
    }

    leaf write-default {
        type action-type;
        default "deny";
        description
            "Controls whether create, update, or delete access
            is granted if no appropriate rule is found for a
            particular write request.";
    }

    leaf exec-default {
        type action-type;
        default "permit";
        description
```

```
    "Controls whether exec access is granted if no appropriate
      rule is found for a particular protocol operation request.";
  }

  leaf enable-external-groups {
    type boolean;
    default true;
    description
      "Controls whether the server uses the groups reported by the
      NETCONF transport layer when it assigns the user to a set of
      NACM groups.  If this leaf has the value 'false', any group
      names reported by the transport layer are ignored by the
      server.";
  }

  leaf denied-operations {
    type yang:zero-based-counter32;
    config false;
    mandatory true;
    description
      "Number of times since the server last restarted that a
      protocol operation request was denied.";
  }

  leaf denied-data-writes {
    type yang:zero-based-counter32;
    config false;
    mandatory true;
    description
      "Number of times since the server last restarted that a
      protocol operation request to alter
      a configuration datastore was denied.";
  }

  leaf denied-notifications {
    type yang:zero-based-counter32;
    config false;
    mandatory true;
    description
      "Number of times since the server last restarted that
      a notification was dropped for a subscription because
      access to the event type was denied.";
  }

  container groups {
    description
      "NETCONF Access Control Groups.";
```

```
list group {
  key name;

  description
    "One NACM Group Entry. This list will only contain
    configured entries, not any entries learned from
    any transport protocols.";

  leaf name {
    type group-name-type;
    description
      "Group name associated with this entry.";
  }

  leaf-list user-name {
    type user-name-type;
    description
      "Each entry identifies the username of
      a member of the group associated with
      this entry.";
  }
}

list rule-list {
  key "name";
  ordered-by user;
  description
    "An ordered collection of access control rules.";

  leaf name {
    type string {
      length "1..max";
    }
    description
      "Arbitrary name assigned to the rule-list.";
  }

  leaf-list group {
    type union {
      type matchall-string-type;
      type group-name-type;
    }
    description
      "List of administrative groups that will be
      assigned the associated access rights
      defined by the 'rule' list.

      The string '*' indicates that all groups apply to the
```

```
        entry.";
    }

    list rule {
        key "name";
        ordered-by user;
        description
            "One access control rule.

            Rules are processed in user-defined order until a match is
            found. A rule matches if 'module-name', 'rule-type', and
            'access-operations' match the request. If a rule
            matches, the 'action' leaf determines if access is granted
            or not.";

        leaf name {
            type string {
                length "1..max";
            }
            description
                "Arbitrary name assigned to the rule.";
        }

        leaf module-name {
            type union {
                type matchall-string-type;
                type string;
            }
            default "*";
            description
                "Name of the module associated with this rule.

                This leaf matches if it has the value '*' or if the
                object being accessed is defined in the module with the
                specified module name.";
        }

        choice rule-type {
            description
                "This choice matches if all leafs present in the rule
                match the request. If no leafs are present, the
                choice matches all requests.";
            case protocol-operation {
                leaf rpc-name {
                    type union {
                        type matchall-string-type;
                        type string;
                    }
                    description
```

```

        "This leaf matches if it has the value '*' or if
        its value equals the requested protocol operation
        name.";
    }
}
case notification {
    leaf notification-name {
        type union {
            type matchall-string-type;
            type string;
        }
        description
            "This leaf matches if it has the value '*' or if its
            value equals the requested notification name.";
    }
}
case data-node {
    leaf path {
        type node-instance-identifier;
        mandatory true;
        description
            "Data Node Instance Identifier associated with the
            data node controlled by this rule.

            Configuration data or state data instance
            identifiers start with a top-level data node. A
            complete instance identifier is required for this
            type of path value.

            The special value '/' refers to all possible
            datastore contents.";
    }
}
}

leaf access-operations {
    type union {
        type matchall-string-type;
        type access-operations-type;
    }
    default "";
    description
        "Access operations associated with this rule.

        This leaf matches if it has the value '*' or if the
        bit corresponding to the requested operation is set.";
}

```

```
    leaf action {
      type action-type;
      mandatory true;
      description
        "The access control action associated with the
         rule.  If a rule is determined to match a
         particular request, then this object is used
         to determine whether to permit or deny the
         request.";
    }

    leaf comment {
      type string;
      description
        "A textual description of the access rule.";
    }
  }
}
}
}

<CODE ENDS>
```

Figure 5

3.6. IANA Considerations

This document registers one URI in "The IETF XML Registry". Following the format in [RFC3688], the following has been registered.

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

This document registers one module in the "YANG Module Names" registry. Following the format in [RFC6020], the following has been registered.

Name: ietf-netconf-acm
Namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-acm
Prefix: nacm
reference: RFC 6536

3.7. Security Considerations

This entire document discusses access control requirements and mechanisms for restricting NETCONF protocol behavior within a given session.

This section highlights the issues for an administrator to consider when configuring a NETCONF server with NACM.

3.7.1. NACM Configuration and Monitoring Considerations

Configuration of the access control system is highly sensitive to system security. A server may choose not to allow any user configuration to some portions of it, such as the global security level or the groups that allowed access to system resources.

By default, NACM enforcement is enabled. By default, "read" access to all datastore contents is enabled (unless "nacm:default-deny-all" is specified for the data definition), and "exec" access is enabled for safe protocol operations. An administrator needs to ensure that NACM is enabled and also decide if the default access parameters are set appropriately. Make sure the following data nodes are properly configured:

- o /nacm/enable-nacm (default "true")
- o /nacm/read-default (default "permit")
- o /nacm/write-default (default "deny")
- o /nacm/exec-default (default "permit")

An administrator needs to restrict write access to all configurable objects within this data model.

If write access is allowed for configuration of access control rules, then care needs to be taken not to disrupt the access control enforcement. For example, if the NACM access control rules are edited directly within the running configuration datastore (i.e., :writable-running capability is supported and used), then care needs to be taken not to allow unintended access while the edits are being done.

An administrator needs to make sure that the translation from a transport- or implementation-dependent user identity to a NACM username is unique and correct. This requirement is specified in detail in Section 2.2 of [RFC6241].

An administrator needs to be aware that the YANG data structures representing access control rules (/nacm/rule-list and /nacm/rule-list/rule) are ordered by the client. The server will evaluate the access control rules according to their relative conceptual order within the running datastore configuration.

Note that the `/nacm/groups` data structure contains the administrative group names used by the server. These group names may be configured locally and/or provided through an external protocol, such as RADIUS [RFC2865][RFC5607].

An administrator needs to be aware of the security properties of any external protocol used by the NETCONF transport layer to determine group names. For example, if this protocol does not protect against man-in-the-middle attacks, an attacker might be able to inject group names that are configured in NACM, so that a user gets more permissions than it should. In such cases, the administrator may wish to disable the usage of such group names, by setting `/nacm/enable-external-groups` to "false".

An administrator needs to restrict read access to the following objects within this data model, as they reveal access control configuration that could be considered sensitive.

- o `/nacm/enable-nacm`
- o `/nacm/read-default`
- o `/nacm/write-default`
- o `/nacm/exec-default`
- o `/nacm/enable-external-groups`
- o `/nacm/groups`
- o `/nacm/rule-list`

3.7.2. General Configuration Issues

There is a risk that invocation of non-standard protocol operations will have undocumented side effects. An administrator needs to construct access control rules such that the configuration datastore is protected from such side effects.

It is possible for a session with some write access (e.g., allowed to invoke `<edit-config>`), but without any access to a particular datastore subtree containing sensitive data, to determine the presence or non-presence of that data. This can be done by repeatedly issuing some sort of edit request (create, update, or delete) and possibly receiving "access-denied" errors in response. These "fishing" attacks can identify the presence or non-presence of specific sensitive data even without the "error-path" field being present within the `<rpc-error>` response.

It may be possible for the set of NETCONF capabilities on the server to change over time. If so, then there is a risk that new protocol operations, notifications, and/or datastore content have been added to the device. An administrator needs to be sure the access control rules are correct for the new content in this case. Mechanisms to detect NETCONF capability changes on a specific device are outside the scope of this document.

It is possible that the data model definition itself (e.g., YANG when-stmt) will help an unauthorized session determine the presence or even value of sensitive data nodes by examining the presence and values of different data nodes.

There is a risk that non-standard protocol operations, or even the standard <get> protocol operation, may return data that "aliases" or "copies" sensitive data from a different data object. There may simply be multiple data model definitions that expose or even configure the same underlying system instrumentation.

A data model may contain external keys (e.g., YANG leafref), which expose values from a different data structure. An administrator needs to be aware of sensitive data models that contain leafref nodes. This entails finding all the leafref objects that "point" at the sensitive data (i.e., "path-stmt" values) that implicitly or explicitly include the sensitive data node.

It is beyond the scope of this document to define access control enforcement procedures for underlying device instrumentation that may exist to support the NETCONF server operation. An administrator can identify each protocol operation that the server provides and decide if it needs any access control applied to it.

This document incorporates the optional use of a recovery session mechanism, which can be used to bypass access control enforcement in emergencies, such as NACM configuration errors that disable all access to the server. The configuration and identification of such a recovery session mechanism are implementation-specific and outside the scope of this document. An administrator needs to be aware of any recovery session mechanisms available on the device and make sure they are used appropriately.

It is possible for a session to disrupt configuration management, even without any write access to the configuration, by locking the datastore. This may be done to ensure all or part of the configuration remains stable while it is being retrieved, or it may be done as a "denial-of-service" attack. There is no way for the server to know the difference. An administrator may wish to restrict "exec" access to the following protocol operations:

- o <lock>
- o <unlock>
- o <partial-lock>
- o <partial-unlock>

3.7.3. Data Model Design Considerations

Designers need to clearly identify any sensitive data, notifications, or protocol operations defined within a YANG module. For such definitions, a "nacm:default-deny-write" or "nacm:default-deny-all" statement ought to be present, in addition to a clear description of the security risks.

Protocol operations need to be properly documented by the data model designer, so it is clear to administrators what data nodes (if any) are affected by the protocol operation and what information (if any) is returned in the <rpc-reply> message.

Data models ought to be designed so that different access levels for input parameters to protocol operations are not required. Use of generic protocol operations should be avoided, and if different access levels are needed, separate protocol operations should be defined instead.

4. References

4.1. Normative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-17 (work in progress), September 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

4.2. Informative References

- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <<http://www.rfc-editor.org/info/rfc2865>>.
- [RFC5607] Nelson, D. and G. Weber, "Remote Authentication Dial-In User Service (RADIUS) Authorization for Network Access Server (NAS) Management", RFC 5607, DOI 10.17487/RFC5607, July 2009, <<http://www.rfc-editor.org/info/rfc5607>>.

Appendix A. Usage Examples

The following XML snippets are provided as examples only, to demonstrate how NACM can be configured to perform some access control tasks.

A.1. <groups> Example

There needs to be at least one <group> entry in order for any of the access control rules to be useful.

The following XML shows arbitrary groups and is not intended to represent any particular use case.

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <groups>
    <group>
      <name>admin</name>
      <user-name>admin</user-name>
      <user-name>andy</user-name>
    </group>

    <group>
      <name>limited</name>
      <user-name>wilma</user-name>
      <user-name>bam-bam</user-name>
    </group>

    <group>
      <name>guest</name>
      <user-name>guest</user-name>
      <user-name>guest@example.com</user-name>
    </group>
  </groups>
</nacm>
```

This example shows three groups:

admin: The "admin" group contains two users named "admin" and "andy".

limited: The "limited" group contains two users named "wilma" and "bam-bam".

guest: The "guest" group contains two users named "guest" and "guest@example.com".

A.2. Module Rule Example

Module rules are used to control access to all the content defined in a specific module. A module rule has the <module-name> leaf set, but no case in the "rule-type" choice.

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <rule-list>
    <name>guest-acl</name>
    <group>guest</group>

    <rule>
      <name>deny-ncm</name>
      <module-name>ietf-netconf-monitoring</module-name>
      <access-operations>*</access-operations>
      <action>deny</action>
      <comment>
        Do not allow guests any access to the NETCONF
        monitoring information.
      </comment>
    </rule>
  </rule-list>

  <rule-list>
    <name>limited-acl</name>
    <group>limited</group>

    <rule>
      <name>permit-ncm</name>
      <module-name>ietf-netconf-monitoring</module-name>
      <access-operations>read</access-operations>
      <action>permit</action>
      <comment>
        Allow read access to the NETCONF
        monitoring information.
      </comment>
    </rule>
    <rule>
      <name>permit-exec</name>
      <module-name>*</module-name>
      <access-operations>exec</access-operations>
      <action>permit</action>
      <comment>
        Allow invocation of the
        supported server operations.
      </comment>
    </rule>
  </rule-list>
</nacm>
```

```
        </comment>
      </rule>
    </rule-list>

    <rule-list>
      <name>admin-acl</name>
      <group>admin</group>

      <rule>
        <name>permit-all</name>
        <module-name>*</module-name>
        <access-operations>*</access-operations>
        <action>permit</action>
        <comment>
          Allow the admin group complete access to all
          operations and data.
        </comment>
      </rule>
    </rule-list>
  </nacm>
```

This example shows four module rules:

deny-ncm: This rule prevents the "guest" group from reading any monitoring information in the "ietf-netconf-monitoring" YANG module.

permit-ncm: This rule allows the "limited" group to read the "ietf-netconf-monitoring" YANG module.

permit-exec: This rule allows the "limited" group to invoke any protocol operation supported by the server.

permit-all: This rule allows the "admin" group complete access to all content in the server. No subsequent rule will match for the "admin" group because of this module rule.

A.3. Protocol Operation Rule Example

Protocol operation rules are used to control access to a specific protocol operation.

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <rule-list>
    <name>guest-limited-acl</name>
```

```
<group>limited</group>
<group>guest</group>

<rule>
  <name>deny-kill-session</name>
  <module-name>ietf-netconf</module-name>
  <rpc-name>kill-session</rpc-name>
  <access-operations>exec</access-operations>
  <action>deny</action>
  <comment>
    Do not allow the limited or guest group
    to kill another session.
  </comment>
</rule>
<rule>
  <name>deny-delete-config</name>
  <module-name>ietf-netconf</module-name>
  <rpc-name>delete-config</rpc-name>
  <access-operations>exec</access-operations>
  <action>deny</action>
  <comment>
    Do not allow limited or guest group
    to delete any configurations.
  </comment>
</rule>
</rule-list>

<rule-list>
  <name>limited-acl</name>
  <group>limited</group>

  <rule>
    <name>permit-edit-config</name>
    <module-name>ietf-netconf</module-name>
    <rpc-name>edit-config</rpc-name>
    <access-operations>exec</access-operations>
    <action>permit</action>
    <comment>
      Allow the limited group to edit the configuration.
    </comment>
  </rule>
</rule-list>

</nacm>
```

This example shows three protocol operation rules:

deny-kill-session: This rule prevents the "limited" or "guest" groups from invoking the NETCONF <kill-session> protocol operation.

deny-delete-config: This rule prevents the "limited" or "guest" groups from invoking the NETCONF <delete-config> protocol operation.

permit-edit-config: This rule allows the "limited" group to invoke the NETCONF <edit-config> protocol operation. This rule will have no real effect unless the "exec-default" leaf is set to "deny".

A.4. Data Node Rule Example

Data node rules are used to control access to specific (config and non-config) data nodes within the NETCONF content provided by the server.

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <rule-list>
    <name>guest-acl</name>
    <group>guest</group>

    <rule>
      <name>deny-nacm</name>
      <path xmlns:n="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
        /n:nacm
      </path>
      <access-operations>*</access-operations>
      <action>deny</action>
      <comment>
        Deny the guest group any access to the /nacm data.
      </comment>
    </rule>
  </rule-list>

  <rule-list>
    <name>limited-acl</name>
    <group>limited</group>

    <rule>
      <name>permit-acme-config</name>
      <path xmlns:acme="http://example.com/ns/netconf">
        /acme:acme-netconf/acme:config-parameters
      </path>
      <access-operations>
        read create update delete
      </access-operations>
```



```
<action>permit</action>
<comment>
  Allow the limited group complete access to the acme
  NETCONF configuration parameters. Showing long form
  of 'access-operations' instead of shorthand.
</comment>
</rule>
</rule-list>

<rule-list>
  <name>guest-limited-acl</name>
  <group>guest</group>
  <group>limited</group>

  <rule>
    <name>permit-dummy-interface</name>
    <path xmlns:acme="http://example.com/ns/itf">
      /acme:interfaces/acme:interface[acme:name='dummy']
    </path>
    <access-operations>read update</access-operations>
    <action>permit</action>
    <comment>
      Allow the limited and guest groups read
      and update access to the dummy interface.
    </comment>
  </rule>
</rule-list>

<rule-list>
  <name>admin-acl</name>
  <group>admin</group>
  <rule>
    <name>permit-interface</name>
    <path xmlns:acme="http://example.com/ns/itf">
      /acme:interfaces/acme:interface
    </path>
    <access-operations>*</access-operations>
    <action>permit</action>
    <comment>
      Allow admin full access to all acme interfaces.
    </comment>
  </rule>
</rule-list>
</nacm>
```

This example shows four data node rules:

deny-nacm: This rule denies the "guest" group any access to the <nacm> subtree. Note that the default namespace is only applicable because this subtree is defined in the same namespace as the <data-rule> element.

permit-acme-config: This rule gives the "limited" group read-write access to the acme <config-parameters>.

permit-dummy-interface: This rule gives the "limited" and "guest" groups read-update access to the acme <interface> entry named "dummy". This entry cannot be created or deleted by these groups, just altered.

permit-interface: This rule gives the "admin" group read-write access to all acme <interface> entries.

A.5. Notification Rule Example

Notification rules are used to control access to a specific notification event type.

```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <rule-list>
    <name>sys-acl</name>
    <group>limited</group>
    <group>guest</group>

    <rule>
      <name>deny-config-change</name>
      <module-name>acme-system</module-name>
      <notification-name>sys-config-change</notification-name>
      <access-operations>read</access-operations>
      <action>deny</action>
      <comment>
        Do not allow the guest or limited groups
        to receive config change events.
      </comment>
    </rule>
  </rule-list>
</nacm>
```

This example shows one notification rule:

deny-config-change: This rule prevents the "limited" or "guest" groups from receiving the acme <sys-config-change> event type.

Authors' Addresses

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

EMail: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

EMail: mbj@tail-f.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks
October 31, 2016

Keystore Model
draft-ietf-netconf-keystore-00

Abstract

This document defines a YANG data module for a system-level keystore mechanism, that might be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

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. No other RFC Editor instructions are specified elsewhere 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:

- o draft-ietf-netconf-restconf
- o draft-ietf-netconf-call-home
- o draft-ietf-rtgwg-yang-key-chain

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

- o "VVVV" --> the assigned RFC value for this draft
- o "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf
- o "YYYY" --> the assigned RFC value for draft-ietf-netconf-call-home

Artwork in this document contains placeholder values for ports pending IANA assignment from "draft-ietf-netconf-call-home". Please apply the following replacements:

- o "7777" --> the assigned port value for "netconf-ch-ssh"
- o "8888" --> the assigned port value for "netconf-ch-tls"
- o "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:

- o "2016-10-31" --> the publication date of this draft

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

- o Appendix A. Change Log
- o Appendix B. 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 May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Requirements Language	4
1.2. Tree Diagram Notation	4
2. The Keystore Model	4
2.1. Overview	5
2.2. Example Usage	6
2.3. YANG Module	17
3. Design Considerations	28
4. Security Considerations	29
5. IANA Considerations	30
5.1. The IETF XML Registry	30
5.2. The YANG Module Names Registry	30
6. Acknowledgements	31
7. References	31
7.1. Normative References	31
7.2. Informative References	32
Appendix A. Change Log	33
A.1. server-model-09 to 00	33
Appendix B. Open Issues	33
Authors' Addresses	33

1. Introduction

This document defines a YANG [RFC6020] data module for a system-level keystore mechanism, which can be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

This module provides a centralized location for security sensitive data, so that the data can be then referenced by other modules. There are two types of data that are maintained by this module:

- o Private keys, and any associated public certificates.
- o Sets of trusted certificates.

This document extends special consideration for systems that have Trusted Protection Modules (TPMs). These systems are unique in that the TPM must be directed to generate new private keys (it is not possible to load a private key into a TPM) and it is not possible to backup/restore the TPM's private keys as configuration.

It is not required that a system has an operating system level keystore utility to implement this module.

1.1. Requirements Language

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 Diagram Notation

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

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

2. The Keystore Model

The keystore module defined in this section provides a configurable object having the following characteristics:

- o A semi-configurable list of private keys, each with one or more associated certificates. Private keys MUST be either preinstalled (e.g., a key associated to an IDevID [Std-802.1AR-2009] certificate), be generated by request, or be loaded by request. Each private key is MAY have associated certificates, either preinstalled or configured after creation.
- o A configurable list of lists of trust anchor certificates. This enables the server to have use-case specific trust anchors. For instance, one list of trust anchors might be used to authenticate management connections (e.g., client certificate-based

authentication for NETCONF or RESTCONF connections), and a different list of trust anchors might be used for when connecting to a specific Internet-based service (e.g., a zero touch bootstrap server).

- o An RPC to generate a certificate signing request for an existing private key, a passed subject, and an optional attributes. The signed certificate returned from an external certificate authority (CA) can be later set using a standard configuration change request (e.g., <edit-config>).
- o An RPC to request the server to generate a new private key using the specified algorithm and key length.
- o An RPC to request the server to load a new private key.

2.1. Overview

The keystore module has the following tree diagram. Please see Section 1.2 for information on how to interpret this diagram.

```

module: ietf-keystore
  +--rw keystore
    +--rw private-keys
      +--rw private-key* [name]
        +--rw name                string
        +--ro algorithm?          identityref
        +--ro key-length?         uint32
        +--ro public-key          binary
        +--rw certificate-chains
          +--rw certificate-chain* [name]
            +--rw name            string
            +--rw certificate*    binary
          +---x generate-certificate-signing-request
            +---w input
              +---w subject      binary
              +---w attributes?  binary
            +--ro output
              +--ro certificate-signing-request  binary
          +---x generate-private-key
            +---w input
              +---w name          string
              +---w algorithm     identityref
              +---w key-length?   uint32
          +---x load-private-key
            +---w input
              +---w name          string
              +---w private-key   binary
  
```



```

+--rw trusted-certificates* [name]
|   +--rw name                string
|   +--rw description?        string
|   +--rw trusted-certificate* [name]
|       +--rw name            string
|       +--rw certificate?    binary
+--rw trusted-ssh-host-keys* [name]
|   +--rw name                string
|   +--rw description?        string
|   +--rw trusted-host-key* [name]
|       +--rw name            string
|       +--rw host-key        binary
+--rw user-auth-credentials
|   +--rw user-auth-credential* [username]
|       +--rw username        string
|       +--rw auth-method* [priority]
|           +--rw priority            uint8
|           +--rw (auth-type)?
|               +--:(certificate)
|                   | +--rw certificate*            -> /keystore/private
-keys/private-key/certificate-chains/certificate-chain/name
|               +--:(public-key)
|                   | +--rw public-key*            -> /keystore/private
-keys/private-key/name
|               +--:(ciphertext-password)
|                   | +--rw ciphertext-password?    string
|               +--:(cleartext-password)
|                   +--rw cleartext-password?        string

notifications:
+---n certificate-expiration
+--ro certificate            instance-identifier
+--ro expiration-date        yang:date-and-time

```

2.2. Example Usage

The following example illustrates the "generate-private-key" action in use with the RESTCONF protocol and JSON encoding.

REQUEST

[\'\' line wrapping added for formatting only]

POST https://example.com/restconf/data/ietf-keystore:keystore/\nprivate-keys/generate-private-key HTTP/1.1\nHOST: example.com\nContent-Type: application/yang.operation+json

```
{
  "ietf-keystore:input" : {
    "name" : "ex-key-sect571r1",
    "algorithm" : "sect571r1"
  }
}
```

RESPONSE

HTTP/1.1 204 No Content\nDate: Mon, 31 Oct 2015 11:01:00 GMT\nServer: example-server

The following example illustrates the "load-private-key" action in use with the RESTCONF protocol and JSON encoding.

REQUEST

['\ ' line wrapping added for formatting only]

POST https://example.com/restconf/data/ietf-keystore:keystore/\nprivate-keys/load-private-key HTTP/1.1\nHOST: example.com\nContent-Type: application/yang.operation+xml

```
<input xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">\n  <name>ex-key-sect571r1</name>\n  <private-key>\n    NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMEdBMVVkRGd\\n\n    VEJiz0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERzd05ER\\n\n    V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF\\n\n    Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\\n\n    QmdOVkhJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\\n\n    MkF6a3hqUDlVQWtHR0dvSlUleUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\\n\n    NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFFVZER3RUIvd1FFQXdJSGdEQnBC\\n\n    WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\\n\n    lLQ1l1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNctadVJMZgprYjk\\n\n    zSFNwSDdwVXBCYnA4dmtNanFtZ jJma3RqZHBxeFppUUtTbndWZTF2Zwot\\n\n    25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXS9RdGp4NULXZmdvN2\\n\n    WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=\n  </private-key>\n</input>
```

RESPONSE

HTTP/1.1 204 No Content\nDate: Mon, 31 Oct 2015 11:01:00 GMT\nServer: example-server

The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol.

REQUEST

```
<rpc message-id="101"\n  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">\n  <action xmlns="urn:ietf:params:xml:ns:yang:1">\n    <keystore\n      xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
```

```

    <private-keys>
      <private-key>
        <name>ex-key-sect571r1</name>
        <generate-certificate-signing-request>
          <subject>
            cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2R
            manZvO3NkZmJpdmhzZGZpbHVidjtvvc2lkZmhidmllbHNlmo
            Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmR6Zgo=
          </subject>
          <attributes>
            bwtakWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvut4
            arnZvO3NkZmJpdmhzZGZpbHVidjtvvc2lkZmhidmllbHNkYm
            Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmC6Rhp=
          </attributes>
        </generate-certificate-signing-request>
      </private-key>
    </private-keys>
  </keystore>
</action>
</rpc>

```

RESPONSE

```

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <certificate-signing-request
    xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
    0F3SUJBZ01kQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWl1ZRFFFQkJRvU
    FNRFF4Q3pBSk1JNlYkQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
    GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApe
    dir1V4RXpBUk1JNlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUV1
    KS29aSWh2Y04KQVFFQk1RQURnWTBtUlHSkFvR0JBTVXVvZmFPNEV3
    El1QWMrQ1RStkNm0d6cEw1Um5ydXZsOFRicUJTdGZQY3N0Zk1KT1
    FaNzlnNlNWVldsMldzaHE1bUViCk1JNnltGNzdjbTAvU25FcFE0TnV
    bXBBD2YkQWdNQkFBR2pnyXZ3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
    URiR01PNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
    ZKY1o2WURiR01PNDB4ajlPb3JtREdsRUNCVTFvVGlrTmPBME1Rc3d
    mMKTTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
    RBR0FRSC9BZ0VBTUEwR0NTcUdTSWl1ZRFFFQgpCUVVBQTRHQAkFMMmx
    rWmFGNWcyAGR6MVNlZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
    TXp4YXJCbFpDSH1LCk1VbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
    c4d0tSSElkyW1WL0pGTmlQS0VXSTF4K1IlaDZmazcrQzQ1QXglRWV
    SWHgZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  </certificate-signing-request>
</rpc-reply>

```

The following example illustrates what a fully configured keystore object might look like. The private-key shown below is consistent with the generate-private-key and generate-certificate-signing-request examples above. This example also assumes that the resulting CA-signed certificate has been configured back onto the server. Lastly, this example shows that three lists of trusted certificates having been configured.

```
<keystore xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">

  <!-- private keys and associated certificates -->
  <private-keys>
    <private-key>
      <name>my-rsa-user-key</name>
      <algorithm>rsa</algorithm>
      <public-key>
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
        mJpdmhZGZpbHVidjtvC2lkZmhidml1bHNkYmZ2aXNiZGZpYmhZG87Zm
        JvO3NkZ25iO29pLmR6Zgo=
      </public-key>
      <certificate-chains>
        <certificate-chain>
          <name>my-rsa-chain</name>
          <certificate>
            ZKYlo2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmPbME1Rc3d
            diRlV4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUv1
            LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
            KS29aSWh2Y04KQVFFQkJRQRnWTBBSU1HSkFvR0JBTVVvZmFPNEV3
            OF3SUJBZ0lKQUptRT2t3bGpNK2pjTUEwR0NTcUdTSWIZRFFFQkJRvU
            FNRFF4Q3pBSkNjTlYkQkFZVEFsVlRNUkF3RGdZRFZRUUtd2R2S0Ud
            GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
            mMKtUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
            RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIZRFFFQgpCUVVBQTRHkFMMmx
            rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
            TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQSlVDeEtFTE40NEY2Zmk2d
            c4d0tSSElkYWlWL0pGTmlQS0VXSTF4K1IlaDZmazcrQzQ1QXglRWV
            SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
          </certificate>
        </certificate-chain>
      </certificate-chains>
    </private-key>

    <private-key>
      <name>my-ec-user-key</name>
      <algorithm>secp256r1</algorithm>
      <public-key>
        mJpdmhZGZpbHVidjtvC2lkZmhidml1bHNkYmZ2aXNiZGZpYmhZG87Zm
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
```

```

    JvO3NkZ25iO29pLmR6Zgo=
  </public-key>
  <certificate-chains>
    <certificate-chain>
      <name>my-ec-chain</name>
      <certificate>
        0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRvU
        ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmPbME1Rc3d
        diRlV4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUV1
        LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
        KS29aSWH2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTVXVvZmFPNEV3
        FNRFF4Q3pBSkNtYkQkFZVEFsVlRNUkF3RGdZRFZRUUtdFd2RsZUd
        GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WAPE
        mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
        RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHkFMMmx
        rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
        TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
        c4d0tSSElkyWlWl0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXglRWV
        SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
      </certificate>
    </certificate-chain>
  </certificate-chains>
</private-key>

<private-key>
  <name>tpm-protected-key</name>
  <algorithm>sect571r1</algorithm>
  <public-key>
    cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
    mJpdmhzZGZpbHVidjtvC2lkZmhidml1bHNkYmZ2aXNiZGZpYmhzZG87Zm
    JvO3NkZ25iO29pLmR6Zgo=
  </public-key>
  <certificate-chains>
    <certificate-chain>
      <name>default-idevid-chain</name>
      <certificate>
        diRlV4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUV1
        LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
        KS29aSWH2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTVXVvZmFPNEV3
        0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRvU
        FNRFF4Q3pBSkNtYkQkFZVEFsVlRNUkF3RGdZRFZRUUtdFd2RsZUd
        GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WAPE
        ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmPbME1Rc3d
        mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
        RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHkFMMmx
        rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
        TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
        c4d0tSSElkyWlWl0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXglRWV

```

```
SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
</certificate>
<certificate>
  KS29aSWH2Y04KQVFFQkJRQURnWTBBTULHSkFvR0JBTXVvZmFPNEV3
  EllQWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJtdGZQY3N0Zk1KT1
  FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAVU25FcFE0TnV
  bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
  0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIZRFFFQkJRvU
  FNRF4Q3pBSk1JnTlYkQkFZVEFsVlRNUkF3RGdZRFZRUUtdFd2RsZUd
  GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApe
  diRlV4RXpBUk1JnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUv1
  URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
  RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIZRFFFQgpCUVVBQTRHkFMMmx
  rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
  c4d0tSSElkyW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXglRWV
  SSUZJQ0FURS0tLS0tCg==
</certificate>
</certificate-chain>
<certificate-chain>
  <name>my-ldev-id-chain</name>
  <certificate>
    0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIZRFFFQkJRvU
    FNRF4Q3pBSk1JnTlYkQkFZVEFsVlRNUkF3RGdZRFZRUUtdFd2RsZUd
    GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApe
    diRlV4RXpBUk1JnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUv1
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
    KS29aSWH2Y04KQVFFQkJRQURnWTBBTULHSkFvR0JBTXVvZmFPNEV3
    EllQWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJtdGZQY3N0Zk1KT1
    FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAVU25FcFE0TnV
    ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmPBME1Rc3d
    mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
    RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIZRFFFQgpCUVVBQTRHkFMMmx
    rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
    TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
    c4d0tSSElkyW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXglRWV
    SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  </certificate>
  <certificate>
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
    0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIZRFFFQkJRvU
    FNRF4Q3pBSk1JnTlYkQkFZVEFsVlRNUkF3RGdZRFZRUUtdFd2RsZUd
    GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApe
    diRlV4RXpBUk1JnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUv1
    KS29aSWH2Y04KQVFFQkJRQURnWTBBTULHSkFvR0JBTXVvZmFPNEV3
    EllQWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJtdGZQY3N0Zk1KT1
    FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAVU25FcFE0TnV
    bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
```

```

    URiR0lPNDB4ajlPb3JtREdsRUNCVTfNR1FHQTFVZApJd1JkTUZ1QU
    ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTfVVGlrTmPBME1Rc3d
    mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
    RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQAkFMMmx
    rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
    TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQSlVDeEtFTE40NEY2Zmk2d
    c4d0tSSElkYWlWL0pGTmlQS0VXSTF4K1IlaDZmazcrQzQ1QXglRWV
    SWHgzZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  </certificate>
</certificate-chain>
</certificate-chains>
</private-key>
</private-keys>

<!-- trusted netconf/restconf client certificates -->
<trusted-certificates>
  <name>explicitly-trusted-client-certs</name>
  <description>
    Specific client authentication certificates that are to be
    explicitly trusted NETCONF/RESTCONF clients. These are
    needed for client certificates not signed by our CA.
  </description>
  <trusted-certificate>
    <name>George Jetson</name>
    <certificate>
      QmdOVKhJBWVRBbFZUTVJBd0RnWURWUWFLRXdkbAplR0Z0Y0d4bE1RNHdEQ
      MkF6a3hqUDlVQWtHR0dvS1UleUclSVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
      25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXS9RdGp4NULXZmdvN2
      RV0JCU2t2MXI2SFNHeUFUVkpwSmYyOWtXbUU0NEo5akJrQmdOVKhTtUUVY
      VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
      UxNQWtHQTFVRUJoTUNWVkl4RURBT0JnTlZCQW9UQjJWNApZVzF3YkdVeE
      V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF
      NQmdOVKhSTUJBZjhFQkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC
      Z05WSFI4RVlqQmdNRjZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN
      WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
      xWVE1SQXdeZ1lEVlFRSwpFd2RsZUdGdGNHeGxNUk13RVFZRFZRUURFd3B
      EVWt3Z1NYTnpkVlZ5TUEwR0NTcUdTSWIzRFFFQkJRUVFBNEdCCkFFc3BK
      WmdsK2gyTTg3QmtGMjhWbWlCdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
      TQzcjFZSjk0M1FQLzV5eGUKN2QxMkxCV0dxUjUrbE15N01YL21ka2M4a1
      zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
      LS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
    </certificate>
  </trusted-certificate>
  <trusted-certificate>
    <name>Fred Flintstone</name>
    <certificate>
      V1EVlFRREV3Vm9ZWEJ3ZVRDQm56QU5CZ2txaGtpRz13MEJBUEUUGQUFPQm
      pRQXdnWWtDCmdZRUE1RzRFSWZsSlp2bDlXTW44eUhyM2hObUFRaUhVUZV

```



```

rRUppQy9hSFA3eGJXQWlra054ZStUa2hrZnBsL3UKbVhsTjhSZUd1ODhG
NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMEdBMVVkRGd
VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNwd4cFJBZ1ZOYUU0cERZd05ER
V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF
NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC
Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN
WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
xWWE1SQXdEZ1lEVlFRSwpFd2RsZUdGdGNHeGxNUk13RVFZRFRZRUURFd3B
EVWt3Z1NYTnpkVlZ5TUEwR0NTcUdTSWlZrFFFFQkJRvUFBNEdCCkFFc3BK
WmdsK2gyTTg3QmtGMjhWbWlCdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
lLQ1l1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNctadVJMZgprYjk
zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBXeFppUUtTbndWZTF2Zwot
QWtUOCBDRVUUZJ0RUF==
</certificate>
</trusted-certificate>
</trusted-certificates>

<!-- trust anchors (CA certs) for netconf/restconf clients -->
<trusted-certificates>
  <name>deployment-specific-ca-certs</name>
  <description>
    Trust anchors used only to authenticate NETCONF/RESTCONF
    client connections. Since our security policy only allows
    authentication for clients having a certificate signed by
    our CA, we only configure its certificate below.
  </description>
  <trusted-certificate>
    <name>ca.example.com</name>
    <certificate>
      WmdsK2gyTTg3QmtGMjhWbWlCdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
      lLQ1l1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNctadVJMZgprYjk
      zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBXeFppUUtTbndWZTF2Zwot
      NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMEdBMVVkRGd
      VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNwd4cFJBZ1ZOYUU0cERZd05ER
      V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF
      NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC
      Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN
      WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
      QmdOVkJBWVRBbFZUTVJBd0RnWURWUvFLRXdkbAp1R0Z0Y0d4bE1RNHdEQ
      MkF6a3hqUDlVQWtHR0dvS1UleUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
      25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXS9RdGp4NULXZmdvN2
      RJSUJQFRStS0Cg==
    </certificate>
  </trusted-certificate>
</trusted-certificates>

<!-- trust anchors for random HTTPS servers on Internet -->
<trusted-certificates>

```

```

<name>common-ca-certs</name>
<description>
  Trusted certificates to authenticate common HTTPS servers.
  These certificates are similar to those that might be
  shipped with a web browser.
</description>
<trusted-certificate>
  <name>ex-certificate-authority</name>
  <certificate>
    NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdZ0VPck1CMEdBMVVkRGd
    VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
    V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF
    Z05WSFI4RVlqQmdNRjZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN
    QmdOVkZBwVRBbFZUTVJBd0RnWURWUWVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ
    MkF6a3hqUDlVQWtHR0dvS1UleUclSVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
    NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC
    WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
    lLQ1l1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNctadVJMZgpRYjk
    zSFNwSDdwVXBCYnA4dmtNanFtZ jJma3RqZHBxeFppUUtTbndWZTF2Zwot
    25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXXZS9RdGp4NULXZmdvN2
    WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
  </certificate>
</trusted-certificate>
</trusted-certificates>

<!-- trusted SSH host keys -->
<trusted-ssh-host-keys>
  <name>explicitly-trusted-ssh-host-keys</name>
  <description>
    Trusted SSH host keys used to authenticate SSH servers.
    These host keys would be analogous to those stored in
    a known_hosts file in OpenSSH.
  </description>
  <trusted-host-key>
    <name>corp-fw1</name>
    <host-key>
      VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
      NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdZ0VPck1CMEdBMVVkRGd
      WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
    </host-key>
  </trusted-host-key>
</trusted-ssh-host-keys>

<!-- user credentials and associated authentication methods -->
<user-auth-credentials>
  <user-auth-credential>
    <username>admin</username>
    <auth-method>

```

```

    <priority>1</priority>
    <certificate-chain>my-ec-chain</certificate-chain>
    <certificate-chain>my-rsa-chain</certificate-chain>
  </auth-method>
  <auth-method>
    <priority>2</priority>
    <public-key>my-rsa-user-key</public-key>
  </auth-method>
</user-auth-credential>
<user-auth-credential>
  <username>tester</username>
  <auth-method>
    <priority>1</priority>
    <cleartext-password>testing123</cleartext-password>
  </auth-method>
</user-auth-credential>
<user-auth-credential>
  <username>ldevid</username>
  <auth-method>
    <priority>1</priority>
    <certificate-chain>my-ldevid-chain</certificate-chain>
  </auth-method>
</user-auth-credential>
</user-auth-credentials>

</keystore>

```

The following example illustrates a "certificate-expiration" notification in XML.

['\`' line wrapping added for formatting only]

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-07-08T00:01:00Z</eventTime>
  <certificate-expiration
    xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
    <certificate>
      /ks:keystore/ks:private-keys/ks:private-key/ks:certificate-chains\
      /ks:certificate-chain/ks:certificate[3]
    </certificate>
    <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
  </certificate-expiration>
</notification>

```

2.3. YANG Module

This YANG module makes extensive use of data types defined in [RFC5280] and [RFC5958].

<CODE BEGINS> file "ietf-keystore@2016-10-31.yang"

```
module ietf-keystore {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keystore";
  prefix "ks";

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

  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 defines a keystore 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 "2016-10-31" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
      Models";
}

identity key-algorithm {
  description
    "Base identity from which all key-algorithms are derived.";
}

identity rsa {
  base key-algorithm;
  description
    "The RSA algorithm.";
  reference
    "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
      RSA Cryptography Specifications Version 2.1.";
}

identity secp192r1 {
  base key-algorithm;
  description
    "The secp192r1 algorithm.";
  reference
    "RFC5480:
      Elliptic Curve Cryptography Subject Public Key Information.";
}

identity secp256r1 {
  base key-algorithm;
  description
    "The secp256r1 algorithm.";
  reference
    "RFC5480:
      Elliptic Curve Cryptography Subject Public Key Information.";
}

identity secp384r1 {
```

```
    base key-algorithm;
    description
      "The secp384r1 algorithm.";
    reference
      "RFC5480:
        Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp521r1 {
    base key-algorithm;
    description
      "The secp521r1 algorithm.";
    reference
      "RFC5480:
        Elliptic Curve Cryptography Subject Public Key Information.";
  }

  container keystore {
    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 keystore.";
      list private-key {
        key name;
        description
          "A private key.";
        leaf name {
          type string;
          description
            "An arbitrary name for the private key.";
        }
        leaf algorithm {
          type identityref {
            base "key-algorithm";
          }
          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 binary;
    config false;
    mandatory true;
    description
        "An OneAsymmetricKey 'publicKey' structure as specified
        by RFC 5958, Section 2 encoded using the ASN.1
        distinguished encoding rules (DER), as specified
        in ITU-T X.690.";
    reference
        "RFC 5958:
        Asymmetric Key Packages
        ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
}
container certificate-chains {
    description
        "Certificate chains associated with this private key.
        More than one chain per key is enabled to support,
        for instance, a TPM-protected key that has associated
        both IDevID and LDevID certificates.";
    list certificate-chain {
        key name;
        description
            "A certificate chain for this public key.";
        leaf name {
            type string;
            description
                "An arbitrary name for the certificate chain. The
                name must be a unique across all private keys, not
                just within this private key.";
        }
        leaf-list certificate {
            type binary;
            ordered-by user;
            description
                "An X.509 v3 certificate structure as specified by RFC
                5280, Section 4 encoded using the ASN.1 distinguished
                encoding rules (DER), as specified in ITU-T X.690.
                The list of certificates that run from the server
                certificate towards the trust anchor. The chain MAY
                include the trust anchor certificate itself.";
            reference
```

```

        "RFC 5280:
          Internet X.509 Public Key Infrastructure Certificate
          and Certificate Revocation List (CRL) Profile.
        ITU-T X.690:
          Information technology - ASN.1 encoding rules:
          Specification of Basic Encoding Rules (BER),
          Canonical Encoding Rules (CER) and Distinguished
          Encoding Rules (DER).";
      }
    }
  }
}
action generate-certificate-signing-request {
  description
    "Generates a certificate signing request structure for
    the associated private key using the passed subject and
    attribute values. Please review both the Security
    Considerations and Design Considerations sections in
    RFC VVVV for more information regarding this action
    statement.";
  input {
    leaf subject {
      type binary;
      mandatory true;
      description
        "The 'subject' field from the CertificationRequestInfo
        structure as specified by RFC 2986, Section 4.1 encoded
        using the ASN.1 distinguished encoding rules (DER), as
        specified in ITU-T X.690.";
      reference
        "RFC 2986:
          PKCS #10: Certification Request Syntax Specification
          Version 1.7.
        ITU-T X.690:
          Information technology - ASN.1 encoding rules:
          Specification of Basic Encoding Rules (BER),
          Canonical Encoding Rules (CER) and Distinguished
          Encoding Rules (DER).";
    }
    leaf attributes {
      type binary;
      description
        "The 'attributes' field from the CertificationRequestInfo
        structure as specified by RFC 2986, Section 4.1 encoded
        using the ASN.1 distinguished encoding rules (DER), as
        specified in ITU-T X.690.";
      reference
        "RFC 2986:
          PKCS #10: Certification Request Syntax Specification

```



```

        Version 1.7.
    ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
    }
}
output {
    leaf certificate-signing-request {
        type binary;
        mandatory true;
        description
            "A CertificationRequest structure as specified by RFC
            2986, Section 4.1 encoded using the ASN.1 distinguished
            encoding rules (DER), as specified in ITU-T X.690.";
        reference
            "RFC 2986:
            PKCS #10: Certification Request Syntax Specification
            Version 1.7.
            ITU-T X.690:
            Information technology - ASN.1 encoding rules:
            Specification of Basic Encoding Rules (BER),
            Canonical Encoding Rules (CER) and Distinguished
            Encoding Rules (DER).";
    }
}
}
}

action generate-private-key {
    description
        "Requests the device to generate 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 /keystore/private-keys. As such, the passed
                value must not match any existing 'name' value.";
        }
        leaf algorithm {
            type identityref {
                base "key-algorithm";
            }
        }
    }
}

```

```
        mandatory true;
        description
            "The algorithm to be used when generating the key.";
    }
    leaf key-length {
        type uint32;
        description
            "For algorithms that need a key length specified
            when generating the key.";
    }
}
}

action load-private-key {
    description
        "Requests the device to load a private key";
    input {
        leaf name {
            type string;
            mandatory true;
            description
                "The name this private-key should have when listed
                in /keystore/private-keys. As such, the passed
                value must not match any existing 'name' value.";
        }
        leaf private-key {
            type binary;
            mandatory true;
            description
                "An OneAsymmetricKey structure as specified by RFC
                5958, Section 2 encoded using the ASN.1 distinguished
                encoding rules (DER), as specified in ITU-T X.690.
                Note that this is the raw private with no shrouding
                to protect it. The strength of this private key
                MUST NOT be greater than the strength of the secure
                connection over which it is communicated. Devices
                SHOULD fail this request if ever that happens.";
            reference
                "RFC 5958:
                 Asymmetric Key Packages
                ITU-T X.690:
                 Information technology - ASN.1 encoding rules:
                 Specification of Basic Encoding Rules (BER),
                 Canonical Encoding Rules (CER) and Distinguished
                 Encoding Rules (DER).";
        }
    }
}
```

```
}  
  
list trusted-certificates {  
  key name;  
  description  
    "A list of trusted certificates. These certificates  
    can be used by a server to authenticate clients, or by clients  
    to authenticate servers. The certificates may be endpoint  
    specific or for certificate authorities (to authenticate many  
    clients at once. Each list of certificates SHOULD be specific  
    to a purpose, as the list as a whole may be referenced by other  
    modules. For instance, a NETCONF server model might point to  
    a list of certificates to use when authenticating client  
    certificates.";  
  leaf name {  
    type string;  
    description  
      "An arbitrary name for this list of trusted certificates.";  
  }  
  leaf description {  
    type string;  
    description  
      "An arbitrary description for this list of trusted  
      certificates.";  
  }  
  list trusted-certificate {  
    key name;  
    description  
      "A trusted certificate for a specific use. Note, this  
      'certificate' is a list in order to encode any  
      associated intermediate certificates.";  
    leaf name {  
      type string;  
      description  
        "An arbitrary name for this trusted certificate. Must  
        be unique across all lists of trusted certificates  
        (not just this list) so that a leafref to it from  
        another module can resolve to unique values.";  
    }  
    leaf certificate { // rename to 'data'?  
      type binary;  
      description  
        "An X.509 v3 certificate structure as specified by RFC  
        5280, Section 4 encoded using the ASN.1 distinguished  
        encoding rules (DER), as specified in ITU-T X.690.";  
      reference  
        "RFC 5280:  
        Internet X.509 Public Key Infrastructure Certificate
```

```

        and Certificate Revocation List (CRL) Profile.
    ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
    }
}

list trusted-ssh-host-keys {
    key name;
    description
        "A list of trusted host-keys. These host-keys can be used
        by clients to authenticate SSH servers. The host-keys are
        endpoint specific. Each list of host-keys SHOULD be
        specific to a purpose, as the list as a whole may be
        referenced by other modules. For instance, a NETCONF
        client model might point to a list of host-keys to use
        when authenticating servers host-keys.";
    leaf name {
        type string;
        description
            "An arbitrary name for this list of trusted SSH host keys.";
    }
    leaf description {
        type string;
        description
            "An arbitrary description for this list of trusted SSH host
            keys.";
    }
}

list trusted-host-key {
    key name;
    description
        "A trusted host key.";
    leaf name {
        type string;
        description
            "An arbitrary name for this trusted host-key. Must be
            unique across all lists of trusted host-keys (not just
            this list) so that a leafref to it from another module
            can resolve to unique values.

```

Note that, for when the SSH client is able to listen for call-home connections as well, there is no reference identifier (e.g., hostname, IP address, etc.) that it can use to uniquely identify the server with. The call-home draft recommends SSH servers use X.509v3

```

        certificates (RFC6187) when calling home.";
    }
    leaf host-key { // rename to 'data'?
        type binary;
        mandatory true;
        description
            "An OneAsymmetricKey 'publicKey' structure as specified
            by RFC 5958, Section 2 encoded using the ASN.1
            distinguished encoding rules (DER), as specified
            in ITU-T X.690.";
        reference
            "RFC 5958:
                Asymmetric Key Packages
            ITU-T X.690:
                Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
                Encoding Rules (DER).";
    }
}
}

/*
Are the auth credentials truly limited to SSH?
Could they be used by an HTTP client to log into an HTTP server?
If truly just for SSH, maybe rename?
*/
container user-auth-credentials {
    description
        "A list of user authentication credentials that can be used
        by an SSH client to log into an SSH server, using any of
        the supported authentication methods (e.g., password,
        public key, client certificate, etc.).";
    list user-auth-credential {
        key username;
        description
            "The authentication credentials for a specific user.";
        leaf username {
            type string;
            description
                "The username of this user. This will be the username
                used, for instance, to log into an SSH server.";
        }
        list auth-method {
            key priority;
            description
                "A method of authenticating as this user.";
            leaf priority {

```

```
    type uint8;
    description
        "When multiple authentication methods in this list are
        supported by the server, the one with the lowest priority
        value will be the one that is used.";
}
choice auth-type {
    description
        "The authentication type.";
    leaf-list certificate {
        type leafref {
            path "/keystore/private-keys/private-key/"
                + "certificate-chains/certificate-chain/name";
        }
        ordered-by user;
        description
            "A list of references to certificates that can be used
            for user authentication. When multiple certificates
            in this list supported by the server, the one that
            comes before the others in the leaf-list will be
            used.";
    }
    leaf-list public-key {
        type leafref {
            path "/keystore/private-keys/private-key/name";
        }
        ordered-by user;
        description
            "A list of references to public keys that can be used
            for user authentication. When multiple public keys
            in this list supported by the server, the one that
            comes before the others in the leaf-list will be
            used.";
    }
    leaf ciphertext-password {
        type string;
        description
            "An ciphertext password. The method of encipherment
            and how that method can be determined from this
            string is implementation-specific.";
    }
    leaf cleartext-password {
        type string;
        description
            "An cleartext password.";
    }
}
}
```

```
    }  
  }  
}  
  
notification certificate-expiration {  
  description  
    "A notification indicating that a configured certificate is  
    either about to expire or has already expired. When to send  
    notifications is an implementation specific decision, but  
    it is RECOMMENDED that a notification be sent once a month  
    for 3 months, then once a week for four weeks, and then once  
    a day thereafter.";  
  leaf certificate {  
    type instance-identifier;  
    mandatory true;  
    description  
      "Identifies which certificate is expiring or is expired.";  
  }  
  leaf expiration-date {  
    type yang:date-and-time;  
    mandatory true;  
    description  
      "Identifies the expiration date on the certificate.";  
  }  
}  
}
```

<CODE ENDS>

3. Design Considerations

This document, along with four other drafts, was split out from the original draft "draft-ietf-netconf-server-model". The split was made so that each draft would have better focus, and also because there was a desire to define client modules, in addition to server modules. The complete list of drafts that resulted from the split includes:

- draft-ietf-netconf-keystore
- draft-ietf-netconf-ssh-client-server
- draft-ietf-netconf-tls-client-server
- draft-ietf-netconf-netconf-client-server
- draft-ietf-netconf-restconf-client-server

This document uses PKCS #10 [RFC2986] for the "generate-certificate-signing-request" action. The use of Certificate Request Message Format (CRMF) [RFC4211] was considered, but it was unclear if there was market demand for it, and so support for CRMF has been left out of this specification. If it is desired to support CRMF in the future, placing a "choice" statement in both the input and output statements, along with an "if-feature" statement on the CRMF option, would enable a backwards compatible solution.

This document puts a limit of the number of elliptical curves supported by default. This was done to match industry trends in IETF best practice (e.g., matching work being done in TLS 1.3). If additional algorithms are needed, they MAY be augmented in by another module, or added directly in a future version of this document.

Both this document and Key Chain YANG Data Model [draft-ietf-rtgwg-yang-key-chain] regard a similar idea. The authors looked at this and agree that they two modules serve different purposes and hence not worth merging into one document. To underscore this further, this document renamed its module from "ietf-keychain" to "ietf-keystore", to contrast it with the other document's module "ietf-key-chain".

For the trusted-certificates list, Trust Anchor Format [RFC5914] was evaluated and deemed inappropriate due to this document's need to also support pinning. That is, pinning a client-certificate to support NETCONF over TLS client authentication.

4. Security Considerations

This document defines a keystore mechanism that is entrusted with the safe keeping of private keys, and the safe keeping of trusted certificates. Nowhere in this API is there an ability to access (read out) a private key once it is known to the keystore. Further, associated public keys and attributes (e.g., algorithm name, key length, etc.) are read-only. That said, this document allows for the deletion of private keys and their certificates, as well the deletion of trusted certificates. Access control mechanisms (e.g., NACM [RFC6536]) MUST be in place so as to authorize such client actions. Further, whilst the data model allows for private keys and trusted certificates in general to be deleted, implementations should be well aware that some private keys (e.g., those in a TPM) and some trusted certificates, should never be deleted, regardless if the authorization mechanisms would generally allow for such actions.

For the "generate-certificate-signing-request" action, it is RECOMMENDED that devices implement assert channel binding [RFC5056], so as to ensure that the application layer that sent the request is

the same as the device authenticated in the secure transport layer was established.

This document defines a data model that includes a list of private keys. These private keys MAY be deleted using standard NETCONF or RESTCONF operations (e.g., <edit-config>). Implementations SHOULD automatically (without explicit request) zeroize these keys in the most secure manner available, so as to prevent the remnants of their persisted storage locations from being analyzed in any meaningful way.

The keystore module define within this document defines the "load-private-key" action enabling a device to load a client-supplied private key. This is a private key with no shrouding to protect it. The strength of this private key MUST NOT be greater than the strength of the underlying secure transport connection over which it is communicated. Devices SHOULD fail this request if ever the strength of the private key is greater then the strength of the underlying transport.

5. IANA Considerations

5.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-keystore
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registration is requested:

name: ietf-keystore
namespace: urn:ietf:params:xml:ns:yang:ietf-keystore
prefix: kc
reference: RFC VVVV

6. 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, Juergen Schoenwaelder; Phil Shafer, Sean Turner, and Bert Wijnen.

7. References

7.1. Normative References

- [draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<http://www.rfc-editor.org/info/rfc2986>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<http://www.rfc-editor.org/info/rfc5958>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.

7.2. Informative References

- [draft-ietf-rtgwg-yang-key-chain]
Lindem, A., Qu, Y., Yeung, D., Chen, I., Zhang, J., and Y. Yang, "Key Chain YANG Data Model", draft-ietf-rtgwg-yang-key-chain (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-ietf-rtgwg-yang-key-chain>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <<http://www.rfc-editor.org/info/rfc4211>>.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <<http://www.rfc-editor.org/info/rfc5056>>.
- [RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <<http://www.rfc-editor.org/info/rfc5914>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [Std-802.1AR-2009]
IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Removed key-usage parameter from generate-private-key action.
- o Now /private-keys/private-key/certificates/certificate/name must be globally unique (unique across all private keys).
- o Added top-level 'trusted-ssh-host-keys' and 'user-auth-credentials' to support SSH client modules.

Appendix B. Open Issues

Please see: <https://github.com/netconf-wg/keystore/issues>.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks
J. Schoenwaelder
Jacobs University Bremen
November 3, 2016

NETCONF Client and Server Models
draft-ietf-netconf-netconf-client-server-01

Abstract

This document defines two YANG modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

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. No other RFC Editor instructions are specified elsewhere 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:

- o draft-ietf-netconf-keystore
- o draft-ietf-netconf-ssh-client-server
- o draft-ietf-netconf-tls-client-server

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

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for draft-ietf-netconf-ssh-client-server
- o "ZZZZ" --> the assigned RFC value for draft-ietf-netconf-tls-client-server

- o "AAAA" --> the assigned RFC value for draft-ietf-netconf-call-home

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

- o "2016-11-02" --> the publication date of this draft

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

- o Appendix A. Change Log
- o Appendix B. 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 May 7, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Tree Diagrams	4
2. The NETCONF Client Model	4
2.1. Tree Diagram	5
2.2. Example Usage	6
2.3. YANG Model	8
3. The NETCONF Server Model	14
3.1. Tree Diagram	15
3.2. Example Usage	17
3.3. YANG Model	21
4. Design Considerations	31
4.1. Support all NETCONF transports	31
4.2. Enable each transport to select which keys to use	32
4.3. Support authenticating NETCONF clients certificates	32
4.4. Support mapping authenticated NETCONF client certificates to usernames	32
4.5. Support both listening for connections and call home	32
4.6. For Call Home connections	32
4.6.1. Support more than one NETCONF client	32
4.6.2. Support NETCONF clients having more than one endpoint	33
4.6.3. Support a reconnection strategy	33
4.6.4. Support both persistent and periodic connections	33
4.6.5. Reconnection strategy for periodic connections	33
4.6.6. Keep-alives for persistent connections	33
4.6.7. Customizations for periodic connections	34
5. Security Considerations	34
6. IANA Considerations	34
6.1. The IETF XML Registry	34
6.2. The YANG Module Names Registry	34
7. Acknowledgements	35
8. References	35
8.1. Normative References	35
8.2. Informative References	36
Appendix A. Change Log	38
A.1. server-model-09 to 00	38
Appendix B. Open Issues	38
Authors' Addresses	38

1. Introduction

This document defines two YANG [RFC6020] modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

NETCONF is defined by [RFC6241]. SSH is defined by [RFC4252], [RFC4253], and [RFC4254]. TLS is defined by [RFC5246]. NETCONF Call Home is defined by [draft-ietf-netconf-call-home]).

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:

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

2. The NETCONF Client Model

The NETCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model supports both the SSH and TLS transport protocols, using the SSH client and TLS client groupings defined in [draft-ietf-netconf-ssh-client-server] and [draft-ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore model defined in [draft-ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF client supports.

2.1. Tree Diagram

Note: all lines are folded at column 71 with no '\ ' character.

```

module: ietf-netconf-client
  +--rw netconf-client
    +--rw initiate {initiate}?
      +--rw netconf-server* [name]
        +--rw name          string
        +--rw (transport)
          +--:(ssh) {ssh-initiate}?
            +--rw ssh
              +--rw address          inet:host
              +--rw port?            inet:port-number
              +--rw server-auth
                | +--rw trusted-ssh-host-keys?  -> /ks:keystore
                | /trusted-ssh-host-keys/name
                | +--rw trusted-ca-certs?      -> /ks:keystore
                | /trusted-certificates/name {ssh-x509-certs}?
                | +--rw trusted-server-certs?  -> /ks:keystore
                | /trusted-certificates/name
                +--rw client-auth
                  +--rw matches* [name]
                    +--rw name          string
                    +--rw match* [name]
                      | +--rw name          string
                      | +--rw trusted-ssh-host-keys?  -> /ks:ke
                      | ystore/trusted-ssh-host-keys/name
                      | +--rw trusted-ca-certs?      -> /ks:ke
                      | ystore/trusted-certificates/name
                      | +--rw trusted-server-certs?  -> /ks:ke
                      | ystore/trusted-certificates/name
                      +--rw user-auth-credentials?  -> /ks:keyst
                      ore/user-auth-credentials/user-auth-credential/username
                  +--rw listen {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 server-auth

```

```

|   +--rw trusted-ssh-host-keys?   -> /ks:keystore
/trusted-ssh-host-keys/name
|   +--rw trusted-ca-certs?        -> /ks:keystore
/trusted-certificates/name {ssh-x509-certs}?
|   +--rw trusted-server-certs?    -> /ks:keystore
/trusted-certificates/name
+--rw client-auth
+--rw matches* [name]
+--rw name                               string
+--rw match* [name]
|   +--rw name                               string
|   +--rw trusted-ssh-host-keys?   -> /ks:ke
ystore/trusted-ssh-host-keys/name
|   +--rw trusted-ca-certs?        -> /ks:ke
ystore/trusted-certificates/name
|   +--rw trusted-server-certs?    -> /ks:ke
ystore/trusted-certificates/name
+--rw user-auth-credentials?   -> /ks:keyst
ore/user-auth-credentials/user-auth-credential/username

```

2.2. Example Usage

The following example illustrates configuring a NETCONF client to initiate connections, using both the SSH and TLS transport protocols, as well as listening for call-home connections, again using both the SSH and TLS transport protocols.

This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

```

<netconf-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-client">

  <!-- NETCONF servers to initiate NETCONF connections to -->
  <initiate>
    <netconf-server>
      <name>corp-fw1</name>
      <ssh>
        <address>corp-fw1.example.com</address>
        <server-auth>
          <trusted-server-certs>
            deployment-specific-ca-certs
          </trusted-server-certs>
        </server-auth>
        <client-auth>
          <matches>
            <match>
              <trusted-ca-certs>

```

```
        deployment-specific-ca-certs
      </trusted-ca-certs>
    </match>
    <user-auth-credentials>Bob</user-auth-credentials>
  </matches>
</client-auth>
</ssh>
</netconf-server>
</initiate>

<!-- endpoints to listen for NETCONF Call Home connections on -->
<listen>
  <endpoint>
    <name>Intranet-facing listener</name>
    <ssh>
      <address>11.22.33.44</address>
      <server-auth>
        <trusted-ca-certs>
          deployment-specific-ca-certs
        </trusted-ca-certs>
        <trusted-server-certs>
          explicitly-trusted-server-certs
        </trusted-server-certs>
        <trusted-ssh-host-keys>
          explicitly-trusted-ssh-host-keys
        </trusted-ssh-host-keys>
      </server-auth>
      <client-auth>
        <matches>
          <match>
            <trusted-ca-certs>
              deployment-specific-ca-certs
            </trusted-ca-certs>
          </match>
          <user-auth-credentials>admin</user-auth-credentials>
        </matches>
        <matches>
          <match>
            <trusted-ca-certs>
              explicitly-trusted-server-certs
            </trusted-ca-certs>
          </match>
          <user-auth-credentials>admin</user-auth-credentials>
        </matches>
        <matches>
          <match>
            <trusted-ca-certs>
              explicitly-trusted-ssh-host-keys
            </trusted-ca-certs>
          </match>
        </matches>
      </client-auth>
    </ssh>
  </endpoint>
</listen>
```

```
        </trusted-ca-certs>
      </match>
      <user-auth-credentials>admin</user-auth-credentials>
    </matches>
  </client-auth>
</ssh>
</endpoint>
</listen>
</netconf-client>
```

2.3. YANG Model

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

```
<CODE BEGINS> file "ietf-netconf-client@2016-11-02.yang"
```

```
module ietf-netconf-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-client";
  prefix "ncc";

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

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-ssh-client {
    prefix ss;
    revision-date 2016-11-02; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  // import ietf-tls-client {
  //   prefix ts;
  //   revision-date 2016-11-02; // stable grouping definitions
  //   reference
  //     "RFC ZZZZ: TLS Client and Server Models";
  // }
```

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>>

Author: Kent Watsen
<<mailto:kwatsen@juniper.net>>

Author: Gary Wu
<<mailto:garywu@cisco.com>>;

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 XXXX; see the RFC itself for full legal notices.";

```
revision "2016-11-02" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: NETCONF Client and Server Models";  
}
```

```
// Features
```

```
feature initiate {  
  description
```

```
    "The 'initiate' feature indicates that the NETCONF client
    supports initiating NETCONF connections to NETCONF servers
    using at least one transport (e.g., SSH, TLS, etc.).";
}

feature ssh-initiate {
  description
    "The 'ssh-initiate' feature indicates that the NETCONF client
    supports initiating SSH connections to NETCONF servers.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature tls-initiate {
  description
    "The 'tls-initiate' feature indicates that the NETCONF client
    supports initiating TLS connections to NETCONF servers.";
  reference
    "RFC 7589: Using the NETCONF Protocol over Transport
    Layer Security (TLS) with Mutual X.509
    Authentication";
}

feature listen {
  description
    "The 'listen' feature indicates that the NETCONF client
    supports opening a port to accept NETCONF server call
    home connections using at least one transport (e.g.,
    SSH, TLS, etc.).";
}

feature ssh-listen {
  description
    "The 'ssh-listen' feature indicates that the NETCONF client
    supports opening a port to listen for incoming NETCONF
    server call-home SSH connections.";
  reference
    "RFC AAAA: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
  description
    "The 'tls-listen' feature indicates that the NETCONF client
    supports opening a port to listen for incoming NETCONF
    server call-home TLS connections.";
  reference
    "RFC AAAA: NETCONF Call Home and RESTCONF Call Home";
}
```

```
container netconf-client {
  description
    "Top-level container for NETCONF client configuration.";

  container initiate {
    if-feature initiate;
    description
      "Configures client initiating underlying TCP connections.";
    list netconf-server {
      key name;
      description
        "List of NETCONF servers the NETCONF client is to initiate
        connections to.";
      leaf name {
        type string;
        description
          "An arbitrary name for the NETCONF server.";
      }
      choice transport {
        mandatory true;
        description
          "Selects between available transports.";
        case ssh {
          if-feature ssh-initiate;
          container ssh {
            description
              "Specifies SSH-specific transport configuration.";
            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
                client will process the IP addresses as if they
                had been explicitly configured in place of the
                hostname.";
            }
            leaf port {
              type inet:port-number;
              default 830;
              description
                "The IP port for this endpoint.  The NETCONF client
                will use the IANA-assigned well-known port if no
                value is specified.";
            }
          }
          uses ss:initiating-ssh-client-grouping;
        }
      }
    }
  }
}
```

```

    }
/*
    case tls {
        if-feature tls-initiate;
        container tls {
            description
                "Specifies TLS-specific transport configuration.";
            uses endpoints-container {
                refine endpoints/endpoint/port {
                    default 6513;
                }
            }
            uses ts:listening-tls-client-grouping {
                augment "client-auth" {
                    description
                        "Augments in the cert-to-name structure.";
                    uses cert-maps-grouping;
                }
            }
        }
    }
*/
}
} // end initiate

container listen {
    if-feature listen;
    description
        "Configures client accepting call-home TCP connections.";
    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 ss:listening-ssh-client-grouping {
                        refine port {
                            default 4334;
                        }
                    }
                }
            }
        }
    }
}
/*
case tls {
    if-feature tls-listen;
    container tls {
        description
            "TLS-specific listening configuration for inbound
            connections.";
        uses ts:listening-tls-client-grouping {
            refine port {
                default 4335;
            }
            augment "client-auth" {
                description
                    "Augments in the cert-to-name structure.";
                uses cert-maps-grouping;
            }
        }
    }
}
*/

```

```
    }  
  }  
} // end listen  
  
grouping cert-maps-grouping {  
  description  
    "A grouping that defines a container around the  
    cert-to-name structure defined in RFC 7407.";  
  container cert-maps {  
    uses x509c2n:cert-to-name;  
    description  
      "The cert-maps container is used by a TLS-based 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 7407: NETCONF over TLS, Section 7";  
  }  
}
```

<CODE ENDS>

3. The NETCONF Server Model

The NETCONF server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

This model also supports both the SSH and TLS transport protocols, using the SSH server and TLS server groupings defined in [draft-ietf-netconf-ssh-client-server] and [draft-ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore model defined in [draft-ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

3.1. Tree Diagram

Note: all lines are folded at column 71 with no '\ ' character.

```

module: ietf-netconf-server
  +--rw netconf-server
    +--rw session-options
    |   +--rw hello-timeout?  uint16
    +--rw listen {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* [name]
    |   |   |   |   |   |   |   +--rw name          string
    |   |   |   |   |   |   |   +--rw (host-key-type)
    |   |   |   |   |   |   |   |   +--:(public-key)
    |   |   |   |   |   |   |   |   |   +--rw public-key?  -> /ks:keystore/
    |   |   |   |   |   |   |   |   |   private-keys/private-key/name
    |   |   |   |   |   |   |   |   |   +--:(certificate)
    |   |   |   |   |   |   |   |   |   |   +--rw certificate?  -> /ks:keystore/
    |   |   |   |   |   |   |   |   |   |   private-keys/private-key/certificate-chains/certificate-chain/name {ssh
    |   |   |   |   |   |   |   |   |   |   -x509-certs}?
    |   |   |   |   |   |   |   |   |   |   |   +--rw client-cert-auth {ssh-x509-certs}?
    |   |   |   |   |   |   |   |   |   |   |   +--rw trusted-ca-certs?      -> /ks:keystore/
    |   |   |   |   |   |   |   |   |   |   |   trusted-certificates/name
    |   |   |   |   |   |   |   |   |   |   |   |   +--rw trusted-client-certs?  -> /ks:keystore/
    |   |   |   |   |   |   |   |   |   |   |   |   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        -> /ks:keystore/private-keys/
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   private-key/certificate-chains/certificate-chain/name
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   +--rw client-auth
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   +--rw trusted-ca-certs?      -> /ks:keystore/
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   trusted-certificates/name
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   +--rw trusted-client-certs?  -> /ks:keystore/
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   trusted-certificates/name
    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   +--rw cert-maps

```

```

|                                     |--rw cert-to-name* [id]
|                                     |--rw id          uint32
|                                     |--rw fingerprint  x509c2n:tls-fingerp
rint
|                                     |--rw map-type     identityref
|                                     |--rw name         string
|--rw call-home {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:host
              |--rw port?      inet:port-number
          |--rw host-keys
            |--rw host-key* [name]
              |--rw name      string
              |--rw (host-key-type)
                |--:(public-key)
                  |--rw public-key?  -> /ks:keystore/
private-keys/private-key/name
|                                     |--:(certificate)
|                                     |--rw certificate?  -> /ks:keystore/
private-keys/private-key/certificate-chains/certificate-chain/name {ssh
-x509-certs}?
|                                     |--rw client-cert-auth {ssh-x509-certs}?
|                                     |--rw trusted-ca-certs?      -> /ks:keystore/
trusted-certificates/name
|                                     |--rw trusted-client-certs?  -> /ks:keystore/
trusted-certificates/name
|                                     |--:(tls) {tls-call-home}?
|                                     |--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      -> /ks:keystore/private-keys/
private-key/certificate-chains/certificate-chain/name
|                                     |--rw client-auth
|                                     |--rw trusted-ca-certs?      -> /ks:keystore/
trusted-certificates/name
|                                     |--rw trusted-client-certs?  -> /ks:keystore/

```

```

trusted-certificates/name
|
|      +--rw cert-maps
|      |      +--rw cert-to-name* [id]
|      |      |      +--rw id          uint32
|      |      |      +--rw fingerprint x509c2n:tls-fingerp
|      |
|      +--rw map-type identityref
|      +--rw name      string
|
+--rw connection-type
|
+--rw (connection-type)?
|
+--:(persistent-connection)
|
|      +--rw persistent!
|      |      +--rw idle-timeout?  uint32
|      |      +--rw keep-alives
|      |      |      +--rw max-wait?      uint16
|      |      |      +--rw max-attempts?  uint8
|      |
|      +--:(periodic-connection)
|      |      +--rw periodic!
|      |      |      +--rw idle-timeout?  uint16
|      |      |      +--rw reconnect_timeout?  uint16
|      |
|      +--rw reconnect-strategy
|      |      +--rw start-with?      enumeration
|      |      +--rw max-attempts?    uint8
|
+--rw reconnect-strategy
|
+--rw start-with?      enumeration
+--rw max-attempts?    uint8

```

3.2. Example Usage

The following example illustrates configuring a NETCONF server to listen for NETCONF client connections using both the SSH and TLS transport protocols, as well as configuring call-home to two NETCONF clients, one using SSH and the other using TLS.

This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

```

<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <listen>

    <!-- listening for SSH connections -->
    <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-key>

```

```
        <certificate>TPM key</certificate>
      </host-key>
    </host-keys>
    <client-cert-auth>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
        explicitly-trusted-client-certs
      </trusted-client-certs>
    </client-cert-auth>
  </ssh>
</endpoint>

<!-- listening for TLS connections -->
<endpoint>
  <name>netconf/tls</name>
  <tls>
    <address>11.22.33.44</address>
    <certificates>
      <certificate>ex-key-sect571r1-cert</certificate>
    </certificates>
    <client-auth>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
        explicitly-trusted-client-certs
      </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>
          <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>
</endpoint>

</listen>
<call-home>
```

```
<!-- calling home to an SSH-based NETCONF client -->
<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>
        <certificate>TPM key</certificate>
      </host-key>
    </host-keys>
    <client-cert-auth>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
        explicitly-trusted-client-certs
      </trusted-client-certs>
    </client-cert-auth>
  </ssh>
  <connection-type>
    <periodic>
      <idle-timeout>300</idle-timeout>
      <reconnect-timeout>60</reconnect-timeout>
    </periodic>
  </connection-type>
  <reconnect-strategy>
    <start-with>last-connected</start-with>
    <max-attempts>3</max-attempts>
  </reconnect-strategy>
</netconf-client>

<!-- calling home to a TLS-based NETCONF client -->
<netconf-client>
  <name>event-correlator</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>
<certificates>
  <certificate>ex-key-sect571r1-cert</certificate>
</certificates>
<client-auth>
  <trusted-ca-certs>
    deployment-specific-ca-certs
  </trusted-ca-certs>
  <trusted-client-certs>
    explicitly-trusted-client-certs
  </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>
      <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>
<connection-type>
  <persistent>
    <idle-timeout>300</idle-timeout>
    <keep-alives>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </keep-alives>
  </persistent>
</connection-type>
<reconnect-strategy>
  <start-with>first-listed</start-with>
  <max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>

</call-home>
</netconf-server>
```


3.3. YANG Model

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

```
<CODE BEGINS> file "ietf-netconf-server@2016-11-02.yang"

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

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
  prefix "ncs";

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

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-ssh-server {
    prefix ss;
    revision-date 2016-11-02; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  import ietf-tls-server {
    prefix ts;
    revision-date 2016-11-02; // stable grouping definitions
    reference
      "RFC ZZZZ: TLS Client and Server Models";
  }

  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 XXXX; see the RFC itself for full legal notices."

```
revision "2016-11-02" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: NETCONF Client and Server Models";  
}
```

// Features

```
feature listen {  
  description  
    "The 'listen' feature indicates that the NETCONF server  
    supports opening a port to accept NETCONF client connections  
    using at least one transport (e.g., SSH, TLS, etc.).";  
}  
  
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 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 7589: Using the NETCONF Protocol over Transport
    Layer Security (TLS) with Mutual X.509
    Authentication";
}

feature call-home {
  description
    "The 'call-home' feature indicates that the NETCONF server
    supports initiating NETCONF call home connections to NETCONF
    clients using at least one transport (e.g., SSH, TLS, etc.).";
  reference
    "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

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-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";
}

// 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 {
  if-feature listen;
  description
    "Configures listen behavior";
  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 ss:listening-ssh-server-grouping {
                    refine port {
                        default 830;
                    }
                }
            }
        }
        case tls {
            if-feature tls-listen;
            container tls {
                description
                    "TLS-specific listening configuration for inbound
                     connections.";
                uses ts:listening-tls-server-grouping {
                    refine port {
                        default 6513;
                    }
                    augment "client-auth" {
                        description
                            "Augments in the cert-to-name structure.";
                        uses cert-maps-grouping;
                    }
                }
            }
        }
    }
}

container call-home {
    if-feature 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 4334;
                }
            }
            uses ss:non-listening-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 4335;
                }
            }
            uses ts:non-listening-tls-server-grouping {
                augment "client-auth" {
                    description
                        "Augments in the cert-to-name structure.";
                    uses cert-maps-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
        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 maximum 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 must close
            the connection when it is ready to release it. Once
            the connection has been closed, the NETCONF server
            will restart its timer until the next connection.";
        leaf idle-timeout {
            type uint16;
            units "seconds";
            default 300; // five minutes
            description
                "Specifies the maximum number of seconds that a
                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
                "Sets 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 an event
        notification message).";
    }
  }
}
container reconnect-strategy {
  description
    "The reconnection strategy directs how a NETCONF server
    reconnects to a NETCONF client, after discovering its
    connection to the client has dropped, 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).";
    }
  }
}

grouping cert-maps-grouping {
  description
    "A grouping that defines a container around the
    cert-to-name structure defined in RFC 7407.";
  container cert-maps {
    uses x509c2n:cert-to-name;
    description
      "The cert-maps container is used by a TLS-based 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 WWW: NETCONF over TLS, Section 7";
  }
}

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.";
    }
}
}
```

<CODE ENDS>

4. Design Considerations

Editorial: this section is a hold over from before, previously called "Objectives". It was only written to support the "server" (not the "client"). The question is if it's better to add the missing "client" parts, or remove this section altogether.

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF client and servers. This scope includes the following objectives:

4.1. Support all NETCONF transports

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

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

4.2. 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.

4.3. Support authenticating NETCONF clients certificates

When a certificate is used to authenticate a NETCONF 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.

4.4. Support mapping authenticated NETCONF client certificates to usernames

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

4.5. Support both listening for connections and call home

The NETCONF 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. 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.

4.6. For Call Home connections

The following objectives only pertain to call home connections.

4.6.1. Support more than one NETCONF client

A NETCONF server may be managed by more than one NETCONF 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.

4.6.2. Support NETCONF clients having more than one endpoint

A NETCONF client managing a NETCONF 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.

4.6.3. Support a reconnection strategy

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

4.6.4. Support both persistent and periodic connections

NETCONF clients may vary greatly on how frequently they need to interact with a NETCONF 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.

4.6.5. 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.

4.6.6. 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 client requirements, and therefore keep-alive settings should be configurable on a per-client basis.

4.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF 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).

5. Security Considerations

A denial of service (DoS) attack MAY occur if the NETCONF server limits the maximum number of NETCONF sessions it will accept (i.e. the 'max-sessions' field in the ietf-netconf-server module is not zero) and either the "hello-timeout" or "idle-timeout" fields in ietf-netconf-server module have been set to indicate the NETCONF server should wait forever (i.e. set to zero).

6. IANA Considerations

6.1. The IETF XML Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

6.2. The YANG Module Names Registry

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

```
name:      ietf-netconf-client
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-client
prefix:    ncc
reference:  RFC XXXX

name:      ietf-netconf-server
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-server
prefix:    ncs
reference:  RFC XXXX
```

7. 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, Sean Turner, 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.

8. References

8.1. Normative References

- [draft-ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), 2016,
<<https://datatracker.ietf.org/html/draft-ietf-netconf-keystore>>.
- [draft-ietf-netconf-ssh-client-server]
Watsen, K., "SSH Client and Server Models", draft-ietf-netconf-ssh-client-server-00 (work in progress), 2016,
<<https://datatracker.ietf.org/html/draft-ietf-netconf-ssh-client-server>>.
- [draft-ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-00 (work in progress), 2016,
<<https://datatracker.ietf.org/html/draft-ietf-netconf-tls-client-server>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <<http://www.rfc-editor.org/info/rfc7407>>.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.

8.2. Informative References

- [draft-ietf-netconf-call-home] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), 2015, <<https://datatracker.ietf.org/html/draft-ietf-netconf-call-home-17>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <<http://www.rfc-editor.org/info/rfc4252>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.

- [RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <<http://www.rfc-editor.org/info/rfc4254>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Added in previously missing ietf-netconf-client module.
- o Added in new features 'listen' and 'call-home' so future transports can be augmented in.

Appendix B. Open Issues

Please see: <https://github.com/netconf-wg/netconf-client-server/issues>.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

A. Gonzalez Prieto
Cisco Systems
A. Clemm
Sympotech
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
S. Chisholm
Ciena
H. Trevino
Cisco Systems
October 31, 2016

NETCONF Support for Event Notifications
draft-ietf-netconf-netconf-event-notifications-01

Abstract

This document defines the support of [event-notifications] by the Network Configuration protocol (NETCONF). [event-notifications] describes capabilities and operations for providing asynchronous message notification delivery. This document discusses how to provide them on top of NETCONF. The capabilities and operations defined between this document and [event-notifications] are intended to obsolete RFC 5277.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Solution Overview	5
2. Solution	5
2.1. Event Streams	6
2.2. Event Stream Discovery	6
2.3. Default Event Stream	9
2.4. Creating a Subscription	9
2.5. Establishing a Subscription	11
2.6. Modifying a Subscription	16
2.7. Deleting a Subscription	21
2.8. Configured Subscriptions	24
2.9. Event (Data Plane) Notifications	33
2.10. Control Plane Notifications	35
3. Backwards Compatibility	44
3.1. Capabilities	44
3.2. Stream Discovery	45
4. Security Considerations	45
5. Acknowledgments	46
6. References	46
6.1. Normative References	46
6.2. Informative References	47
Appendix A. Issues that are currently being worked	47
Appendix B. Changes between revisions	47
B.1. v00 to v01	47
Authors' Addresses	47

1. Introduction

[RFC6241] can be conceptually partitioned into four layers:

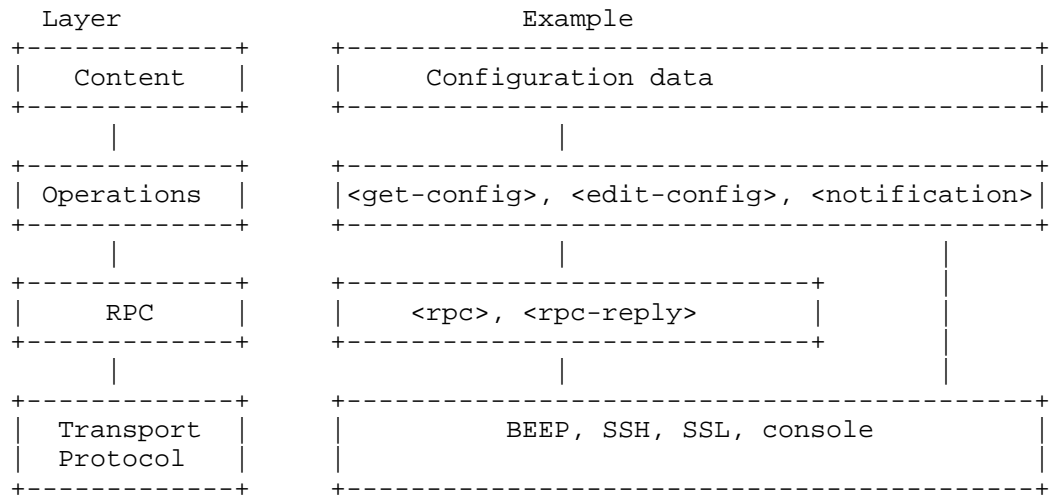


Figure 1: NETCONF layer architecture

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol [RFC6241] based on [event-notifications]. This is an optional capability built on top of the base NETCONF definition.

[event-notifications] and this document enhance the capabilities of RFC 5277 while maintaining backwards capability with existing implementations. It is intended that a final version of this document might obsolete [RFC5277]. The enhancements include the ability to terminate subscriptions without terminating the client session, to modify existing subscriptions, and to have multiple subscriptions on a NETCONF session. [RFC5277] clients that do not require these enhancements are not affected by them.

[event-notifications] covers the following functionality:

- o Ability to subscribe to event notifications using two mechanisms: dynamic and configuration subscriptions.
- o Ability to subscribe to event notifications using two mechanisms: dynamic and configuration subscriptions.
- o Ability to negotiate acceptable subscription parameters.

- o Ability to filter the subset of notifications to be pushed with stream-specific semantics.
- o Ability to support multiple encodings for the notification.
- o Mechanism to communicate the notifications.
- o Ability to replay locally logged notifications.

To support this functionality, NETCONF agents must implement the operations, configuration and operational state defined in [event-notifications]. In addition, they need to:

- o support multiple subscriptions over a single NETCONF session.
- o support a revised definition of the default NETCONF stream
- o be backwards compatible with RFC 5277 implementations.

1.1. Terminology

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

1.1.1. NETCONF

The following terms are defined in [RFC6241] :

- o Client
- o Server
- o Operation
- o RPC: remote procedure call

1.1.2. Event Notifications

The following terms are defined in [event-notifications]:

- o Event
- o Event notification
- o Stream (also referred to as "event stream")
- o Subscriber

- o Publisher
- o Receiver
- o Subscription
- o Filter
- o Dynamic subscription
- o Configured subscription

Note that a publisher in [event-notifications] corresponds to a server in [RFC6241]. Similarly, a subscribers corresponds to a client. A receiver is also a client. In the remainder of this document, we will use the terminology in [RFC6241].

1.1.3. NETCONF Access Control

The following terms are defined in [RFC6536]:

- o NACM: NETCONF Access Control Model

1.2. Solution Overview

[event-notifications] defines mechanisms that provide an asynchronous message notification delivery service. This document discusses its realization on top of the NETCONF protocol [RFC6241].

The functionality to support is defined in [event-notifications]. It is formalized in a set of yang models. The mapping of yang constructs into NETCONF is described in [RFC6020].

Supporting [event-notifications] requires enhancements and modifications in NETCONF. The key enhancement is supporting multiple subscriptions on a NETCONF session. A key modification is the definition of the NETCONF stream.

These enhancements do not affect [RFC5277] clients that do not support [event-notifications].

2. Solution

In this section, we describe and exemplify how [event-notifications] must be supported over NETCONF.

2.1. Event Streams

In the context of NETCONF, an event stream is a set of events available for subscription from a NETCONF server. It is out of the scope of this document to identify a) how streams are defined, b) how events are defined/generated, and c) how events are assigned to streams.

The following is a high-level description of the flow of a notification. Note that it does not mandate and/or preclude an implementation. As events are raised, they are assigned to streams. An event may be assigned to multiple streams. The event is distributed to subscribers and receivers based on the current subscriptions and access control. Access control is needed because if any receiver of that subscription does not have permission to receive an event, then it never makes it into a notification, and processing of the event is completed for that subscription.

2.2. Event Stream Discovery

A NETCONF client can retrieve the list of available event streams from a NETCONF server using the <get> operation. The reply contains the elements defined in the YANG model under the container "/streams", which includes the stream identifier.

The following example illustrates the retrieval of the list of available event streams using the <get> operation.

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <streams
        xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
      </filter>
    </get>
  </rpc>
```

Figure 2: Get streams

The NETCONF server returns a list of event streams available for subscription. In this example, the list contains the NETCONF, SNMP, and syslog-critical streams.


```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <streams
      xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications">
      <stream>NETCONF</stream>
      <stream>SNMP</stream>
      <stream>syslog-critical</stream>
      <stream>NETCONF</stream>
    </streams>
  </data>
</rpc-reply>
```

Figure 3: Get streams response

2.2.1. Backwards Compatibility

In order to maintain backwards compatibility, clients that only support [RFC5277] can retrieve the list of available event streams executing a <get> operation against the container "/netconf/streams".

The following example illustrates this mechanism.

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <netconf
        xmlns="urn:ietf:params:xml:ns:netmod:notification">
        <streams/>
      </netconf>
    </filter>
  </get>
</rpc>
```

Figure 4: Get streams (backwards compatibility)

The NETCONF server returns a list of event streams available for subscription. In this example, the list contains the NETCONF, SNMP, and syslog-critical streams.

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <netconf
      xmlns="urn:ietf:params:xml:ns:netmod:notification">
      <streams>
        <stream>
```

```
<name>
  NETCONF
</name>
<description>
  default NETCONF event stream
</description>
<replaySupport>
  true
</replaySupport>
<replayLogCreationTime>
  2016-02-05T00:00:00Z
</replayLogCreationTime>
</stream>
<stream>
  <name>
    SNMP
  </name>
  <description>
    SNMP notifications
  </description>
  <replaySupport>
    false
  </replaySupport>
</stream>
<stream>
  <name>
    syslog-critical
  </name>
  <description>
    Critical and higher severity
  </description>
  <replaySupport>
    true
  </replaySupport>
  <replayLogCreationTime>
    2007-07-01T00:00:00Z
  </replayLogCreationTime>
</stream>
</streams>
</netconf>
</data>
</rpc-reply>
```

Figure 5: Get streams response (backwards compatibility)

2.3. Default Event Stream

A NETCONF server implementation supporting the notification capability MUST support the "NETCONF" notification event stream. This stream contains all NETCONF XML event notifications supported by the NETCONF server, except for those belonging only to streams that explicitly indicate that they must be excluded from the NETCONF stream. The exact string "NETCONF" is used during the advertisement of stream support during the <get> operation on <streams> and during the <create-subscription> and <establish-subscription> operations.

2.4. Creating a Subscription

This operation was fully defined in [RFC5277].

2.4.1. Usage Example

The following demonstrates dynamically creating a subscription.

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    </create-subscription>
  </netconf:rpc>
```

Figure 6: Create subscription

2.4.2. Positive Response

If the NETCONF server can satisfy the request, the server sends an <ok> element.

```
<netconf:rpc netconf:message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/event/1.0"
      select="/ex:event[ex:eventClass='fault' and
        (ex:severity='minor' or ex:severity='major'
          or ex:severity='critical')]" />
    </create-subscription>
  </netconf:rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 7: Successful create subscription

2.4.3. Negative Response

If the request cannot be completed for any reason, an `<rpc-error>` element is included within the `<rpc-reply>`. Subscription requests can fail for several reasons including if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

If a `stopTime` is specified in a request without having specified a `startTime`, the following error is returned:

```
Tag: missing-element
Error-type: protocol
Severity: error
Error-info: <bad-element>: startTime
Description: An expected element is missing.
```

Figure 8: Create subscription missing an element

If the optional replay feature is requested but the NETCONF server does not support it, the following error is returned:

Tag: operation-failed
Error-type: protocol
Severity: error
Error-info: none
Description: Request could not be completed because the
 requested operation failed for some reason
 not covered by any other error condition.

Figure 9: Create subscription operation failed

If a stopTime is requested that is earlier than the specified
startTime, the following error is returned:

Tag: bad-element
Error-type: protocol
Severity: error
Error-info: <bad-element>: stopTime
Description: An element value is not correct;
 e.g., wrong type, out of range, pattern mismatch.

Figure 10: Create subscription incorrect stopTime

If a startTime is requested that is later than the current time, the
following error is returned:

Tag: bad-element
Error-type: protocol
Severity: error
Error-info: <bad-element>: startTime
Description: An element value is not correct;
 e.g., wrong type, out of range, pattern mismatch.

Figure 11: Create subscription incorrect startTime

2.5. Establishing a Subscription

This operation is defined in [event-notifications].

2.5.1. Usage Example

The following illustrates the establishment of a simple subscription.

```

<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
  </establish-subscription>
</netconf:rpc>

```

Figure 12: Establish subscription

2.5.2. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive <subscription-result> element, and the subscription-id of the accepted subscription.

```

<netconf:rpc netconf:message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/event/1.0"
      select="/ex:event[ex:eventClass='fault' and
        (ex:severity='minor' or ex:severity='major'
        or ex:severity='critical')]" />
    </establish-subscription>
  </netconf:rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    ok
  </subscription-result>
  <subscription-id
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    52
  </subscription-id>
</rpc-reply>

```

Figure 13: Successful establish-subscription

2.5.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends a negative <subscription-result> element.

If the client has no authorization to establish the subscription, the `<subscription-result>` indicates an authorization error. For instance:

```
<netconf:rpc netconf:message-id="103"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <stream>foo</stream>
  </establish-subscription>
</netconf:rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    error-data-not-authorized
  </subscription-result>
</rpc-reply>
```

Figure 14: Unsuccessful establish subscription

If the request is rejected because the server is not able to serve it, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request when it was processed. However, there is no guarantee that subsequent requests with those parameters for this client or others will be accepted. For instance, consider a subscription from [yang-push], which augments the `establish-subscription` with some additional parameters, including "period". If the client requests a period the NETCONF server cannot serve, the exchange may be:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      500
    </period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    error-insufficient-resources
  </subscription-result>
  <period
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    2000
  </period>
</rpc-reply>
```

Figure 15: Subscription establishment negotiation

Subscription requests will fail if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

2.5.4. Multiple Subscriptions over a Single NETCONF Session

Note that [event-notifications] requires supporting multiple subscription establishments over a single NETCONF session. In contrast, [RFC5277] mandated servers to return an error when a create-subscription was sent while a subscription was active on that session. Note that servers are not required to support multiple create-subscription over a single session, but they MUST support multiple establish-subscription over one session.

2.5.5. Message Flow Examples

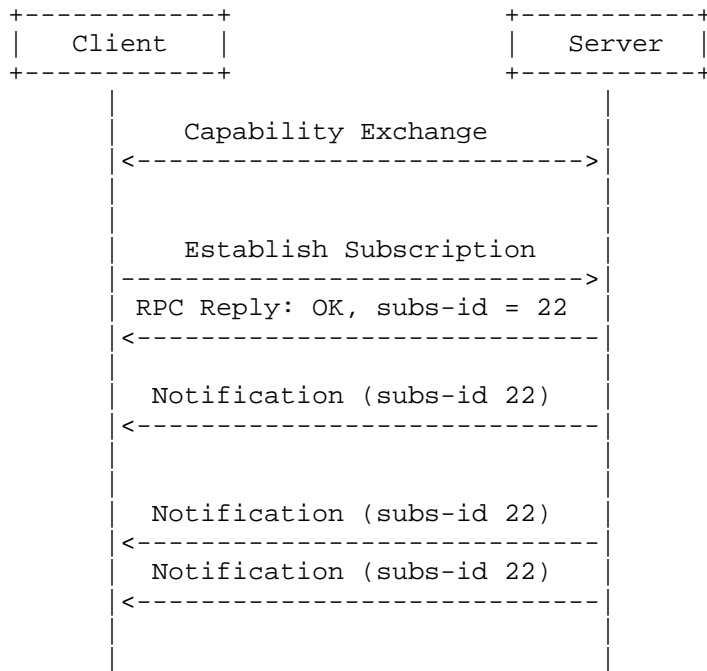


Figure 16: Message flow for subscription establishment

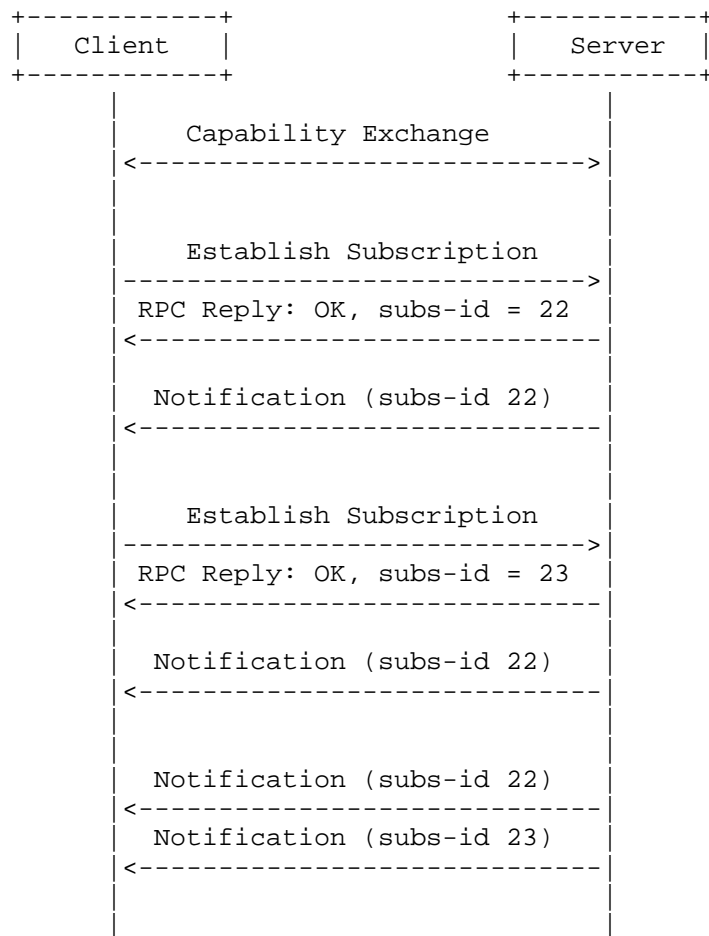


Figure 17: Message Flow for multiple subscription establishments over a single session

2.6. Modifying a Subscription

This operation is defined in [event-notifications].

2.6.1. Usage Example

The following demonstrates modifying a subscription. Consider a subscription from [yang-push], which augments the establish-subscription with some additional parameters, including "period". A subscription may be established as follows.

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      500
    </period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    ok
  </subscription-result>
  <subscription-id
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    1922
  </subscription-id>
</rpc-reply>
```

Figure 18: Establish subscription to be modified

The subscription may be modified with:

```
<netconf:rpc message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <subscription-id>1922</subscription-id>
    <period>1000</period>
  </modify-subscription >
</netconf:rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    ok
  </subscription-result>
  <subscription-id
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    1922
  </subscription-id>
</rpc-reply>
```

Figure 19: Modify subscription

2.6.2. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive `<subscription-result>` element. This response is like that to an establish-subscription request, but without the subscription-id (which would be redundant).

```
<netconf:rpc message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <subscription-id>1922</subscription-id>
    <period
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      1000
    </period>
  </modify-subscription >
</netconf:rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    ok
  </subscription-result>
</rpc-reply>
```

Figure 20: Successful modify subscription

2.6.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends a negative `<subscription-result>` element. Its contents and semantics are identical to those in an establish-subscription request. For instance:

```
<netconf:rpc message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <subscription-id>1922</subscription-id>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      100
    </period>
  </modify-subscription>
</netconf:rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    error-insufficient-resources
  </subscription-result>
  <period>500</period>
</rpc-reply>
```

Figure 21: Unsuccessful modify subscription

2.6.4. Message Flow Example

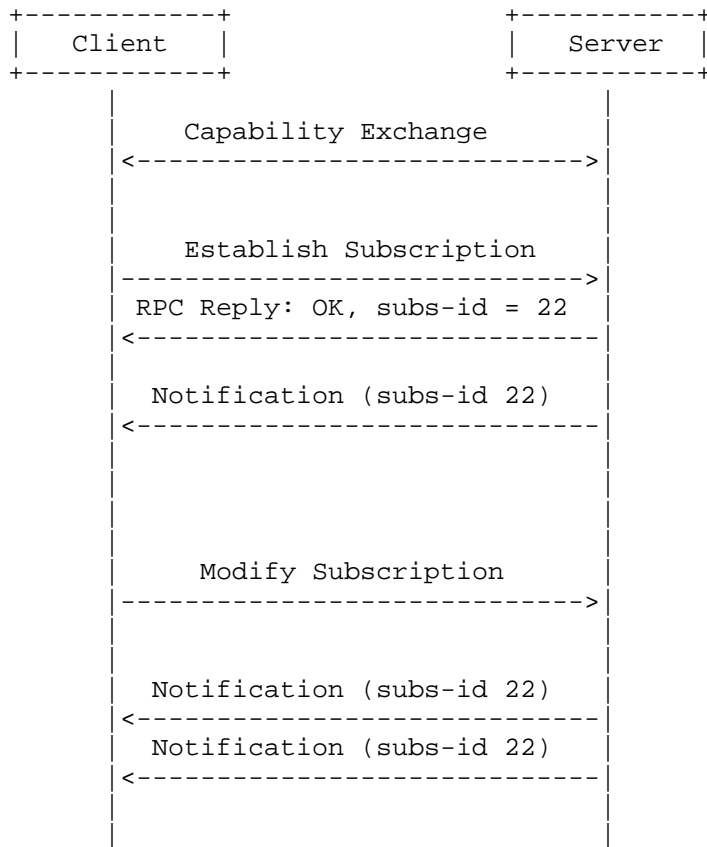


Figure 22: Message flow for subscription modification

2.7. Deleting a Subscription

This operation is defined in [event-notifications].

2.7.1. Usage Example

The following demonstrates deleting a subscription.

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <subscription-id>1922</subscription-id>
  </delete-subscription>
</netconf:rpc>
```

Figure 23: Delete subscription

2.7.2. Positive Response

If the NETCONF server can satisfy the request, the server sends an OK element. For example:

```
<netconf:rpc message-id="103"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <subscription-id>1922</subscription-id>
  </delete-subscription>
</netconf:rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 24: Successful delete subscription

2.7.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends an error-rpc element. For example:


```
<netconf:rpc message-id="103"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
    <subscription-id>2017</subscription-id>
  </delete-subscription>
</netconf:rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path
      xmlns:t="urn:ietf:params:xml:ns:netconf:notification:1.1">
      /t:subscription-id
    </error-path>
    <error-message xml:lang="en">
      Subscription-id 2017 does not exist
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 25: Unsuccessful delete subscription

2.7.4. Message Flow Example

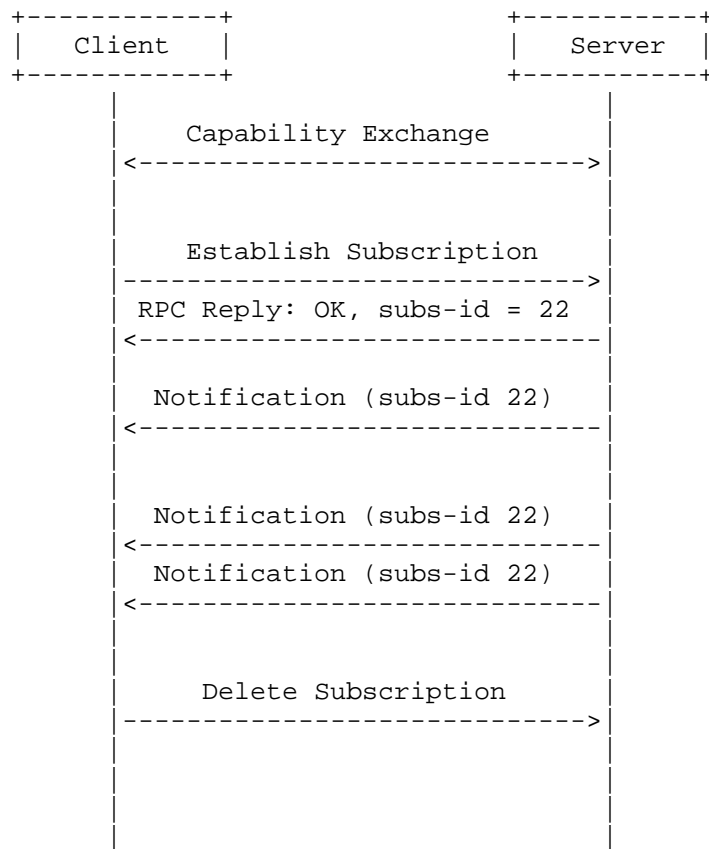


Figure 26: Message flow for subscription deletion

2.8. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface. Configured subscriptions do not support negotiation.

Supporting configured subscriptions is optional and advertised during the capabilities exchange using the "configured-subscriptions" feature.

Configured subscriptions are supported by NETCONF servers using NETCONF Call Home [call-home]

In this section, we present examples of how to manage configuration subscriptions using a NETCONF client.

2.8.1. Call Home for Configured Subscriptions

Configured subscriptions are established, modified, and deleted using configuration operations against the top-level subtree subscription-config. Once the configuration is set, the server initiates a Call Home to each of the receivers in the subscription on the address and port specified. Once the NETCONF session between the server and the receiver is established, the server will issue a "subscription-started" notification. After that, the server will send notifications to the receiver as per the subscription notification.

Note that the server assumes the receiver is aware that calls on the configured port are intended only for pushing notifications. It also assumes that the receiver is ready to accept notifications on the session created as part of the Call Home as soon as the NETCONF session is established. This may require coordination between the client that configures the subscription and the clients for which the notifications are intended. This coordination is out of the scope of this document.

2.8.2. Establishing a Configured Subscription

Subscriptions are established using configuration operations against the top-level subtree subscription-config.

For example at subscription establishment, a NETCONF client may send:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
      <subscription>
        <subscription-id>
          1922
        </subscription-id>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.4
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 27: Establish static subscription

if the request is accepted, the server would reply:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 28: Response to a successful static subscription establishment

if the request is not accepted because the server cannot serve it,
the server may reply:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>resource-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">
      Temporarily the server cannot serve this
      subscription due to the current workload.
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 29: Response to a failed static subscription establishment

2.8.2.1. Message Flow Example

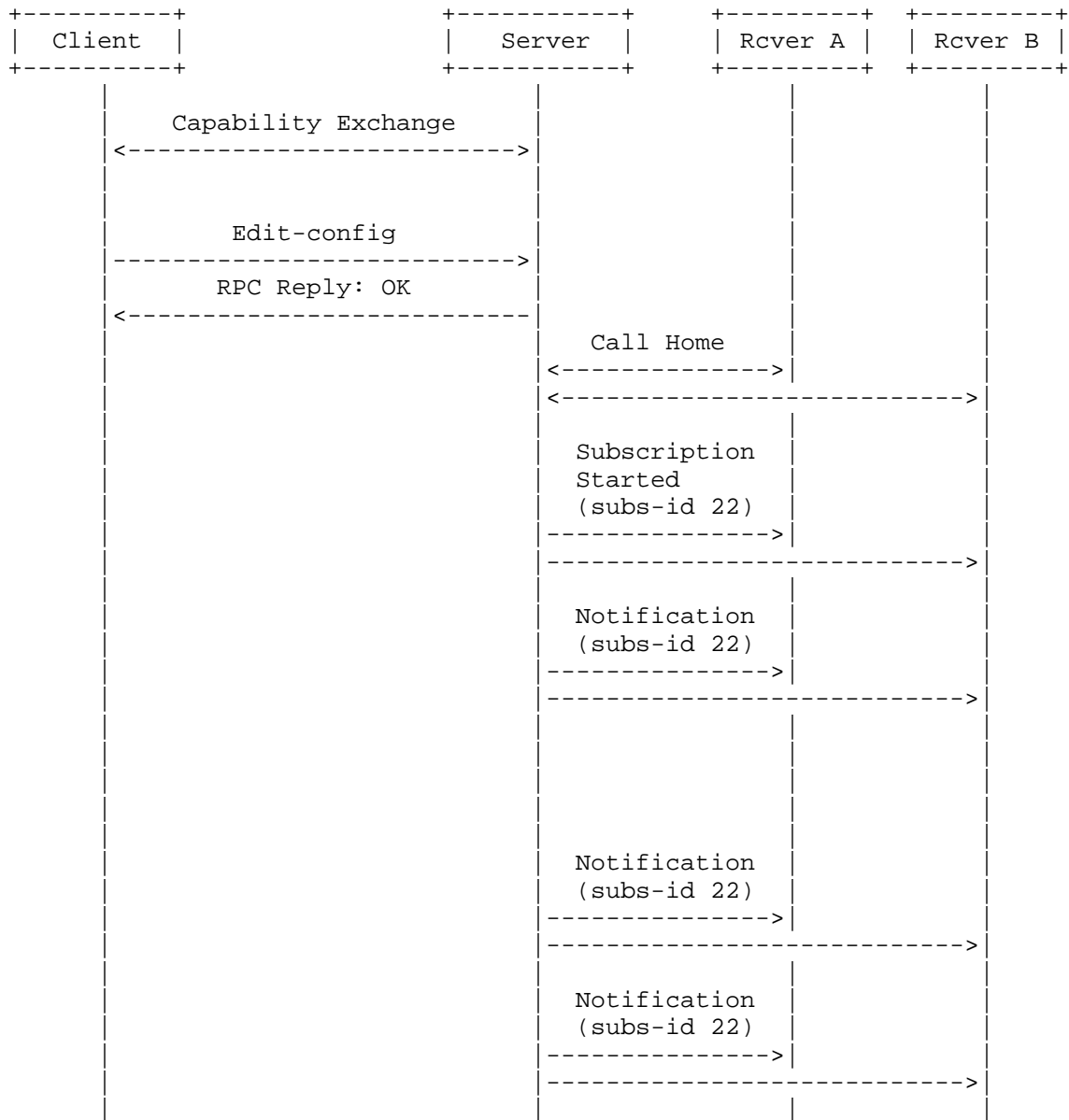


Figure 30: Message flow for subscription establishment (configured subscription)

2.8.3. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level subtree subscription-config.

For example, the subscription established in the previous section could be modified as follows, choosing a different receiver:

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
      <subscription>
        <subscription-id>
          1922
        </subscription-id>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.5
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 31: Modify configured subscription

if the request is accepted, the server would reply:

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 32: Response to a successful configured subscription modification

2.8.3.1. Message Flow Example

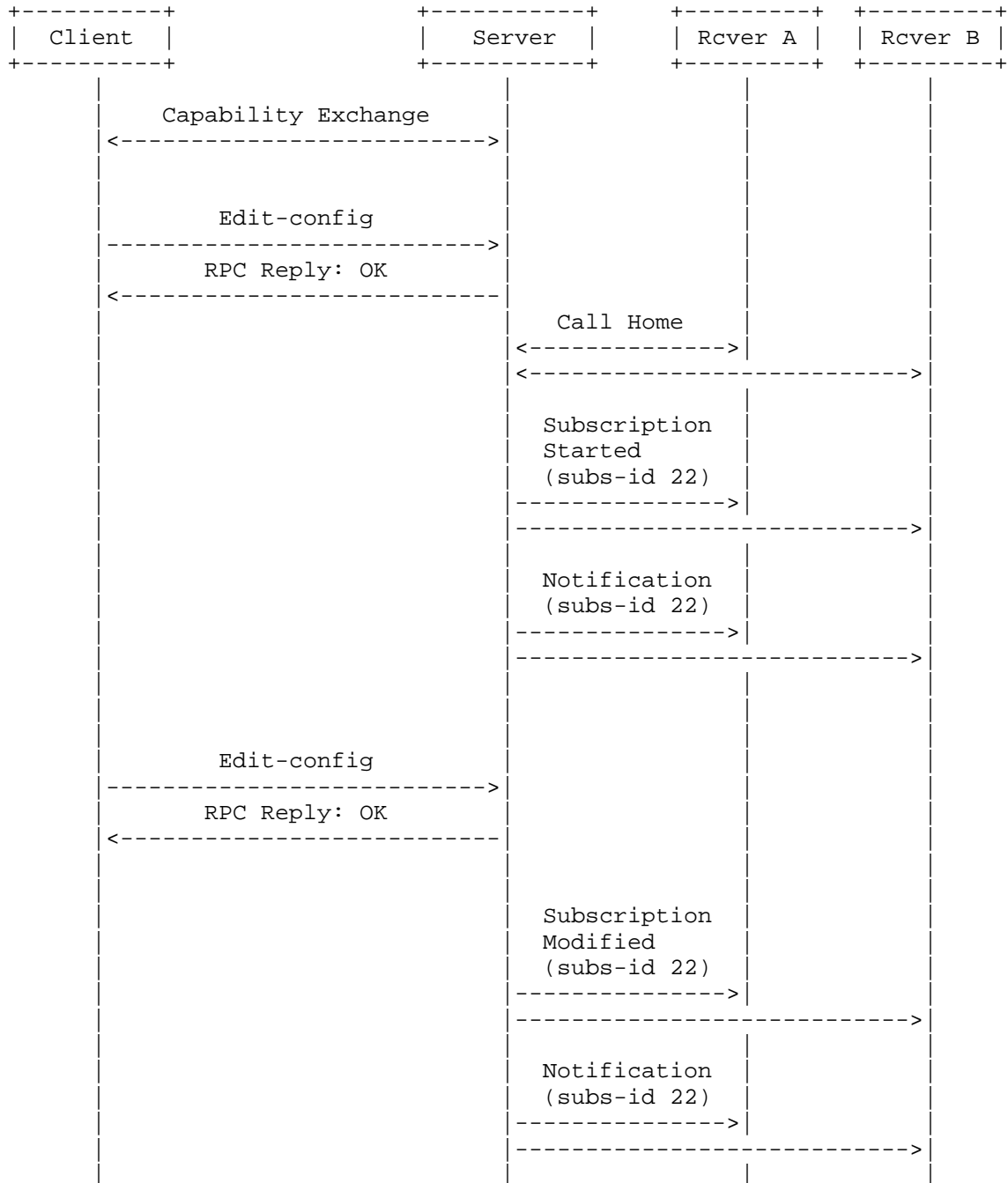


Figure 33: Message flow for subscription modification (configured subscription)

2.8.4. Deleting a Configured Subscription

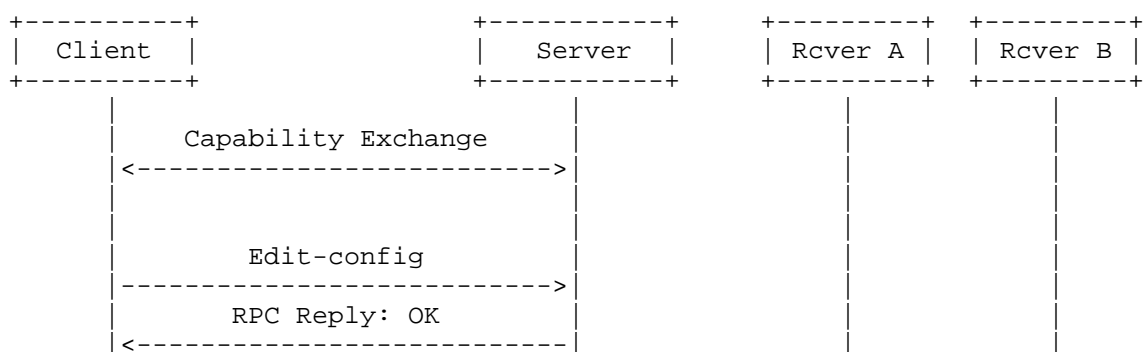
Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example:

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns:xc="urn:ietf:params:xml:ns:netconf:notification:1.1">
      <subscription xc:operation="delete">
        <subscription-id>
          1922
        </subscription-id>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 34: Deleting a configured subscription

2.8.4.1. Message Flow Example



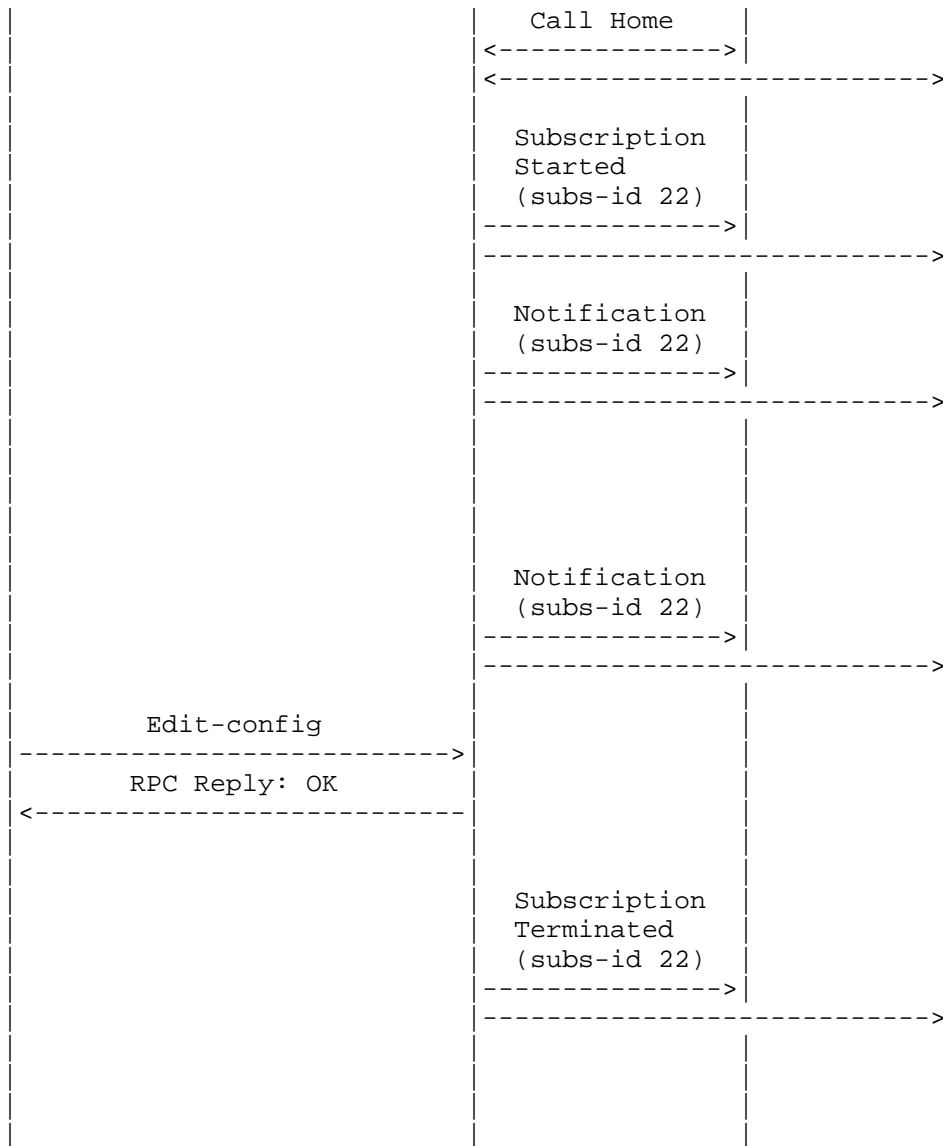


Figure 35: Message flow for subscription deletion (configured subscription)

2.9. Event (Data Plane) Notifications

Once a subscription has been set up, the NETCONF server sends (asynchronously) the event notifications from the subscribed stream. We refer to these as data plane notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the NETCONF session used to create or establish the subscription. For static subscriptions, event notifications are sent over the specified connections.

An event notification is sent to the receiver(s) when an event of interest (i.e., meeting the specified filtering criteria) has occurred. An event notification is a complete and well-formed XML document. Note that `<notification>` is not a Remote Procedure Call (RPC) method but rather the top-level element identifying the one-way message as a notification. Note that event notifications never trigger responses.

The event notification always includes an `<eventTime>` element. It is the time the event was generated by the event source. This parameter is of type `dateTime` and compliant to [RFC3339]. Implementations must support time zones.

The event notification also contains notification-specific tagged content, if any. With the exception of `<eventTime>`, the content of the notification is beyond the scope of this document.

For encodings other than XML, notifications include an additional XML element so that the notification is a well-formed XML. The element is `<notification-contents-{encoding}>`, E.g., `<notification-contents-json>`. That element contains the notification contents in the desired encoding

The following is an example of an event notification from [RFC6020]:

```

notification link-failure {
  description "A link failure has been detected";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}

```

Figure 36: Definition of a data plane notification

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>

```

Figure 37: Data plane notification

The equivalent using JSON encoding would be

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
    {
      "acme-system:link-failure": {
        "if-name": "so-1/2/3.0",
        "if-admin-status": "up",
        "if-oper-status": "down"
      }
    }
  </notification-contents-json>
</notification>

```

Figure 38: Data plane notification using JSON encoding

2.10. Control Plane Notifications

In addition to data plane notifications, a server may send control plane notifications (defined in [event-notifications]) to indicate to receivers that an event related to the subscription management has occurred. Control plane notifications cannot be filtered out. Next we exemplify them using both XML, and JSON encodings for the notification-specific content:

2.10.1. replayComplete

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <replayComplete xmlns="urn:ietf:params:xml:ns:netmod:notification"/>
</notification>
```

Figure 39: replayComplete control plane notification

The equivalent using JSON encoding would be:

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
  {
    "netmod-notif:replayComplete": { }
  }
  </notification-contents-json>
</notification>
```

Figure 40: replayComplete control plane notification (JSON encoding)

2.10.1.1. Message Flow Example

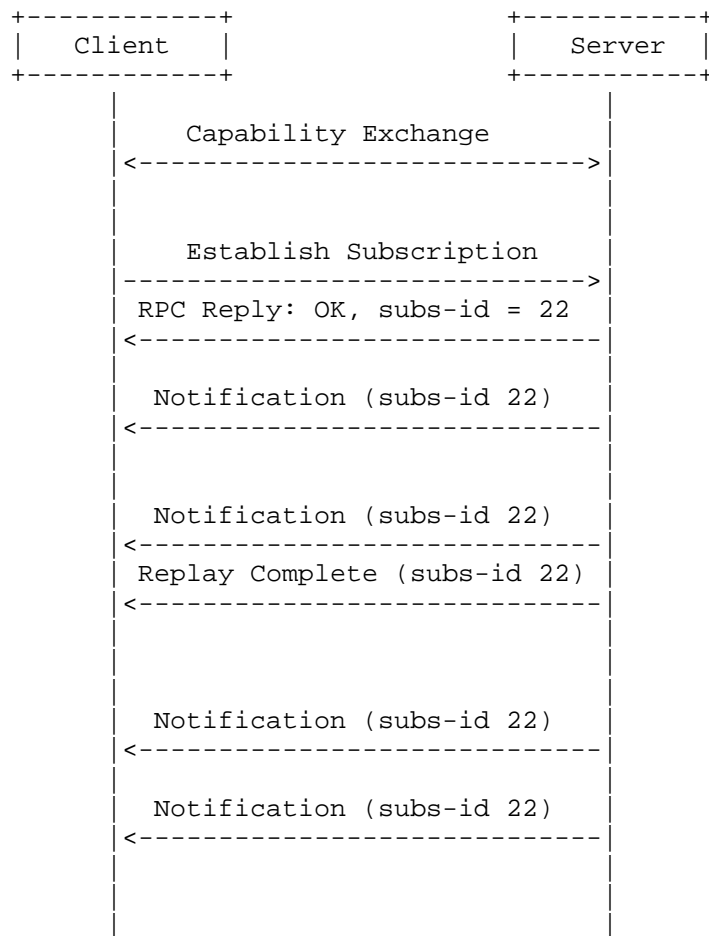


Figure 41: replayComplete notification

2.10.2. notificationComplete

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notificationComplete
    xmlns="urn:ietf:params:xml:ns:netmod:notification"/>
</notification>

```

Figure 42: notificationComplete control plane notification

2.10.2.1. Message Flow Example

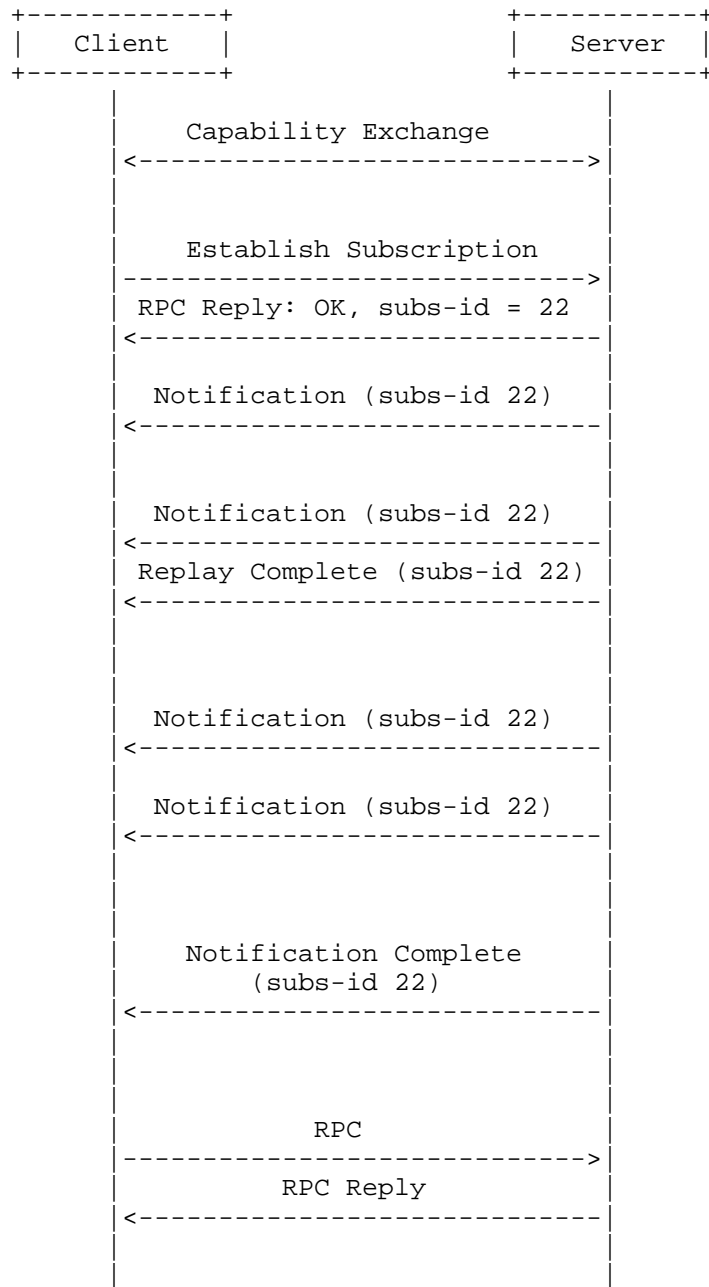


Figure 43: notificationComplete notification

2.10.3. subscription-started

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-started
    xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
    <subscription-id>52</subscription-id>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/event/1.0"
      select="/ex:event[ex:eventClass='fault' and
        (ex:severity='minor' or ex:severity='major'
          or ex:severity='critical')]">
    </subscription-started/>
  </notification>

```

Figure 44: subscription-started control plane notification

The equivalent using JSON encoding would be:

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
    {
      "notif-bis:subscription-started": {
        "subscription-id" : 52
        ((Open Item: express filter in json))
      }
    }
  </notification-contents-json>
</notification>

```

Figure 45: subscription-started control plane notification (JSON encoding)

2.10.4. subscription-modified

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-modified
    xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
    <subscription-id>52</subscription-id>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/event/1.0"
      select="/ex:event[ex:eventClass='fault']"/>
  </subscription-modified/>
</notification>
```

Figure 46: subscription-modified control plane notification

2.10.5. subscription-terminated

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-terminated
    xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
    <subscription-id>52</subscription-id>
    <reason>subscription-deleted</reason>
  </subscription-terminated/>
</notification>
```

Figure 47: subscription-terminated control plane notification

2.10.6. subscription-suspended

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-suspended
    xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
    <subscription-id>52</subscription-id>
    <reason>internal-error</reason>
  </subscription-suspended/>
</notification>
```

Figure 48: subscription-suspended control plane notification

2.10.7. subscription-resumed

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-resumed
    xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
    <subscription-id>52</subscription-id>
    <reason>internal-error</reason>
  </subscription-resumed/>
</notification>
```

Figure 49: subscription-resumed control plane notification

2.10.7.1. Message Flow Examples

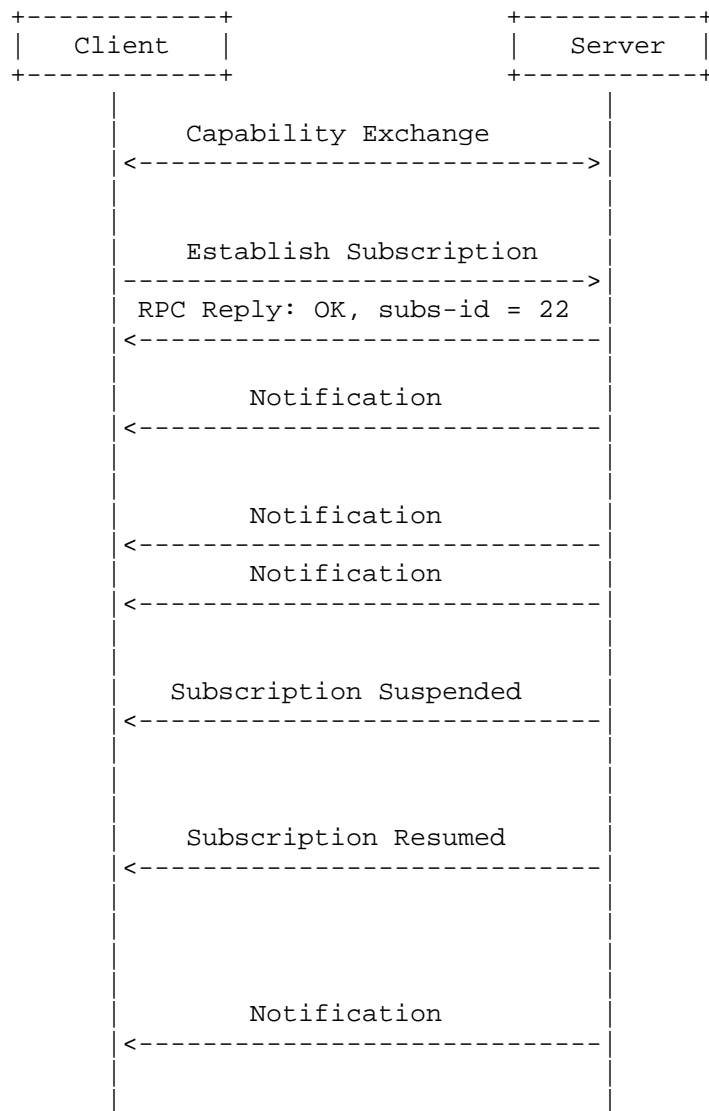
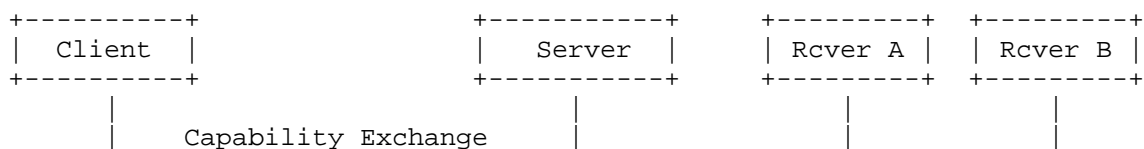


Figure 50: subscription-suspended and Resumed Notifications



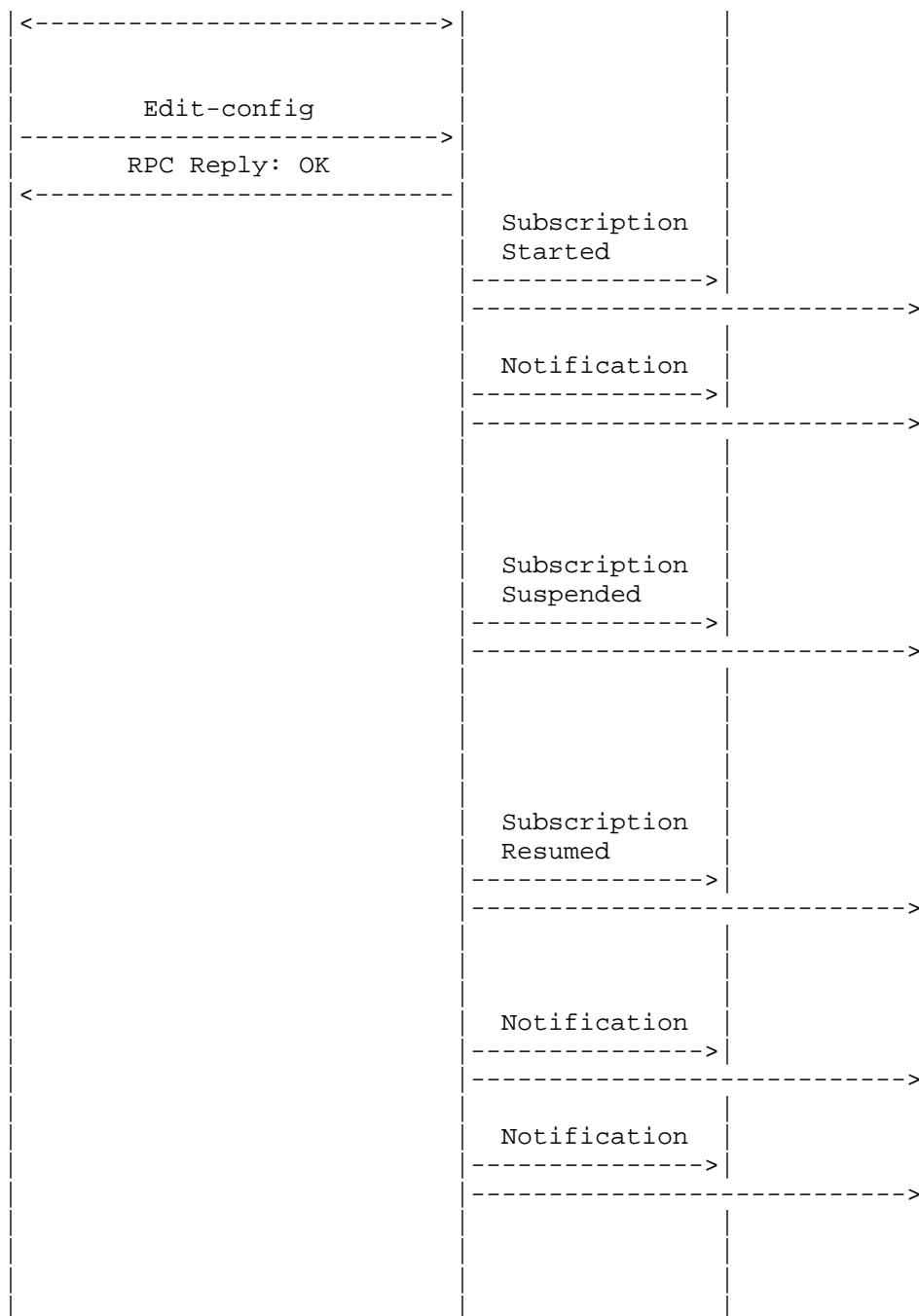


Figure 51: subscription-suspended and subscription-resumed notifications (configured subscriptions)

3. Backwards Compatibility

3.1. Capabilities

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Servers supporting the features in this document must advertise both capabilities "urn:ietf:params:netconf:capability:notification:1.0" and "urn:ietf:params:netconf:capability:notification:1.1".

An example of a hello message by a server during session establishment would be:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:xml:ns:netconf:base:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:startup:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:notification:1.0
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:notification:1.1
    </capability>
  </capabilities>
  <session-id>4</session-id>
</hello>
```

Figure 52: Hello message

Clients that only support [RFC5277] recognize capability "urn:ietf:params:netconf:capability:notification:1.0" and ignore capability "urn:ietf:params:netconf:capability:notification:1.1". This allows them interacting with the server as per [RFC5277]. Clients that support the features in this document recognize both capabilities. This allows them interacting with the server as per this document.

3.2. Stream Discovery

In order to maintain backwards compatibility, clients that only support [RFC5277] can retrieve the list of available event streams executing a <get> operation against the container `"/netconf/streams"`.

4. Security Considerations

The security considerations from the base NETCONF document [RFC6241] also apply to the notification capability.

The <notification> elements are never sent before the transport layer and the NETCONF layer, including capabilities exchange, have been established and the manager has been identified and authenticated.

A secure transport must be used and the server must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of NETCONF content involved. When a <get> is received that refers to the content defined in this memo, clients should only be able to view the content for which they have sufficient privileges. <create-subscription> and <establish-subscription> operations can be considered like deferred <get>, and the content that different users can access may vary. This different access is reflected in the <notification> that different users are able to subscribe to.

The contents of notifications, as well as the names of event streams, may contain sensitive information and care should be taken to ensure that they are viewed only by authorized users. The NETCONF server MUST NOT include any content in a notification that the user is not authorized to view.

If a malicious or buggy NETCONF client sends a number of <create-subscription> requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions can be terminated by terminating the suspect underlying NETCONF sessions using the <kill-session> operation. If the client uses <establish-subscription>, the server can also suspend or terminate subscriptions with per-subscription granularity.

A subscription could be configured on another receiver's behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver, which doesn't even support subscriptions. Clients that do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the NETCONF Authorization Control Model [RFC6536] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user

permissions to receive specific event notification types. The permissions are specified as a set of access control rules.

Note that streams can define additional authorization requirements. For instance, in [yang-push] each of the elements in its data plane notifications must also go through access control.

5. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Yan Gang, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

6.2. Informative References

[call-home]

Watsen, K., "NETCONF Call Home and RESTCONF Call Home", December 2015, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-call-home/>>.

[event-notifications]

Clemm, A., Gonzalez Prieto, A., Voit, Eric., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to Event Notifications", June 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-rfc5277bis/>>.

[yang-push]

Clemm, A., Gonzalez Prieto, A., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", February 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

Appendix A. Issues that are currently being worked

(To be removed by RFC editor prior to publication)

- o NT1 - Express filter in JSON should be documented.

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

B.1. v00 to v01

- o D1 - Added Call Home in solution for configured subscriptions.
- o D2 - Clarified support for multiple subscription on a single session. No need to support multiple create-subscription.
- o D3 - Added mapping between terminology in [yang-push] and [RFC6241] (the one followed in this document).
- o D4 - Editorial improvements.

Authors' Addresses

Alberto Gonzalez Prieto
Cisco Systems
Email: albertgo@cisco.com

Alexander Clemm
Sympotech
Email: alex@sympotech.com

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Einar Nilsen-Nygaard
Cisco Systems
Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems
Email: ambtripa@cisco.com

Sharon Chisholm
Ciena
Email: schishol@ciena.com

Hector Trevino
Cisco Systems
Email: htrevino@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

K. Watsen
Juniper Networks
J. Schoenwaelder
Jacobs University Bremen
November 3, 2016

RESTCONF Client and Server Models
draft-ietf-netconf-restconf-client-server-01

Abstract

This document defines two YANG modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server. Both modules support the TLS transport protocol with both standard RESTCONF and RESTCONF Call Home connections.

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. No other RFC Editor instructions are specified elsewhere 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:

- o draft-ietf-netconf-keystore
- o draft-ietf-netconf-tls-client-server

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

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for draft-ietf-netconf-restconf
- o "ZZZZ" --> the assigned RFC value for draft-ietf-netconf-tls-client-server

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

- o "2016-11-02" --> the publication date of this draft

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

- o Appendix A. Change Log
- o Appendix B. 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 May 7, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Tree Diagrams	3
2. The RESTCONF Client Model	4
2.1. Tree Diagram	4
2.2. Example Usage	4
2.3. YANG Model	4

3.	The RESTCONF Server Model	7
3.1.	Tree Diagram	7
3.2.	Example Usage	9
3.3.	YANG Model	11
4.	Security Considerations	20
5.	IANA Considerations	20
5.1.	The IETF XML Registry	20
5.2.	The YANG Module Names Registry	20
6.	Acknowledgements	20
7.	References	21
7.1.	Normative References	21
7.2.	Informative References	21
Appendix A.	Change Log	23
A.1.	server-model-09 to 00	23
Appendix B.	Open Issues	23
Authors' Addresses	23

1. Introduction

This document defines two YANG [RFC6020] modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server [draft-ietf-netconf-restconf]. Both modules support the TLS [RFC5246] transport protocol with both standard RESTCONF and RESTCONF Call Home connections [draft-ietf-netconf-call-home].

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:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. The RESTCONF Client Model

EDITOR NOTE: Please ignore this section, it is incomplete.

The RESTCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model supports both TLS transport protocols using the TLS client groupings defined in [draft-ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keystore model defined in [draft-ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF client supports.

2.1. Tree Diagram

Note: all lines are folded at column 71 with no '\ ' character.

```
module: ietf-restconf-client
  +--rw restconf-client
    +--rw initiate {tls-initiate}?
    +--rw listen {tls-listen}?
```

2.2. Example Usage

The following example illustrates configuring a RESTCONF client to initiate connections, as well as listening for call-home connections.

This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

FIXME

2.3. YANG Model

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

<CODE BEGINS> file "ietf-restconf-client@2016-11-02.yang"

```
// Editor's Note:
// This module is incomplete at this time.  Below is
// just a skeleton so there's something in the draft.
// Please ignore this module for now!

module ietf-restconf-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-client";
  prefix "rcc";

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

  //import ietf-netconf-acm {
  //  prefix nacm;
  //  reference
  //    "RFC 6536: Network Configuration Protocol (NETCONF)
  //      Access Control Model";
  //}

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-tls-client {
    prefix ts;
    revision-date 2016-11-02; // stable grouping definitions
    reference
      "RFC ZZZZ: TLS Client and Server Models";
  }
*/
  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 XXXX; see the RFC itself for full legal notices.";

```
revision "2016-11-02" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: RESTCONF Client and Server Models";  
}
```

```
// Features
```

```
feature tls-initiate {  
  description  
    "The tls-initiate feature indicates that the RESTCONF client  
    supports initiating TLS connections to RESTCONF servers.";  
  reference  
    "RFC YYYY: RESTCONF Protocol";  
}
```

```
feature tls-listen {  
  description  
    "The tls-listen feature indicates that the RESTCONF client  
    supports opening a port to listen for incoming RESTCONF  
    server call-home TLS connections.";  
  reference  
    "RFC AAAA: NETCONF Call Home and RESTCONF Call Home";  
}
```



```
    }

    container restconf-client {
      description
        "Top-level container for RESTCONF client configuration.";
      container initiate {
        if-feature tls-initiate;
        description
          "Configures client initiating underlying TCP connections.";
      }
      container listen {
        if-feature tls-listen;
        description
          "Configures client accepting call-home TCP connections.";
      }
    }
  }
}
```

<CODE ENDS>

3. The RESTCONF Server Model

The RESTCONF Server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

This model supports the TLS using the TLS server groupings defined in [draft-ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keystore model defined in [draft-ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF server supports.

3.1. Tree Diagram

Note: all lines are folded at column 71 with no '\ ' character.

```
module: ietf-restconf-server
  +--rw restconf-server
    +--rw listen {listen}?
      |   +--rw max-sessions?  uint16
      |   +--rw endpoint* [name]
      |       +--rw name      string
      |       +--rw (transport)
```

```

    +---:(tls) {tls-listen}?
    +---rw tls
    +---rw address?          inet:ip-address
    +---rw port?             inet:port-number
    +---rw certificates
    |   +---rw certificate* [name]
    |   |   +---rw name      -> /ks:keystore/private-keys/
private-key/certificate-chains/certificate-chain/name
    |   +---rw client-auth
    |   +---rw trusted-ca-certs?      -> /ks:keystore/
trusted-certificates/name
    |   +---rw trusted-client-certs?  -> /ks:keystore/
trusted-certificates/name
    |   +---rw cert-maps
    |   |   +---rw cert-to-name* [id]
    |   |   +---rw id              uint32
    |   |   +---rw fingerprint     x509c2n:tls-fingerp
rint
    |   |   +---rw map-type         identityref
    |   |   +---rw name             string
    +---rw call-home {call-home}?
    +---rw restconf-client* [name]
    |   +---rw name                 string
    +---rw (transport)
    |   +---:(tls) {tls-call-home}?
    |   +---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      -> /ks:keystore/private-keys/
private-key/certificate-chains/certificate-chain/name
    |   |   +---rw client-auth
    |   |   +---rw trusted-ca-certs?      -> /ks:keystore/
trusted-certificates/name
    |   |   +---rw trusted-client-certs?  -> /ks:keystore/
trusted-certificates/name
    |   |   +---rw cert-maps
    |   |   |   +---rw cert-to-name* [id]
    |   |   |   +---rw id              uint32
    |   |   |   +---rw fingerprint     x509c2n:tls-fingerp
rint
    |   |   +---rw map-type         identityref
    |   |   +---rw name             string
    +---rw connection-type

```

```

|   +--rw (connection-type)?
|   |   +--:(persistent-connection)
|   |   |   +--rw persistent!
|   |   |   |   +--rw keep-alives
|   |   |   |   |   +--rw max-wait?      uint16
|   |   |   |   |   +--rw max-attempts?  uint8
|   |   +--:(periodic-connection)
|   |   |   +--rw periodic!
|   |   |   |   +--rw reconnect-timeout?  uint16
+--rw reconnect-strategy
|   +--rw start-with?      enumeration
|   +--rw max-attempts?    uint8

```

3.2. Example Usage

The following example illustrates configuring a RESTCONF server to listen for RESTCONF client connections, as well as configuring call-home to one RESTCONF client.

This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

```

<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">

  <!-- listening for TLS (HTTPS) connections -->
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>ex-key-sect571r1-cert</certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>
            deployment-specific-ca-certs
          </trusted-ca-certs>
          <trusted-client-certs>
            explicitly-trusted-client-certs
          </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>
        <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>

</endpoint>
</listen>

<!-- calling home to a RESTCONF client -->
<call-home>
  <restconf-client>
    <name>config-manager</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>
      <certificates>
        <certificate>ex-key-sect571r1-cert</certificate>
      </certificates>
      <client-auth>
        <trusted-ca-certs>
          deployment-specific-ca-certs
        </trusted-ca-certs>
        <trusted-client-certs>
          explicitly-trusted-client-certs
        </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>
            <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>
  <connection-type>
    <periodic>
      <idle-timeout>300</idle-timeout>
      <reconnect-timeout>60</reconnect-timeout>
    </periodic>
  </connection-type>
  <reconnect-strategy>
    <start-with>last-connected</start-with>
    <max-attempts>3</max-attempts>
  </reconnect-strategy>
</restconf-client>
</call-home>

</restconf-server>
```

3.3. YANG Model

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

```
<CODE BEGINS> file "ietf-restconf-server@2016-11-02.yang"

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

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
  prefix "rcs";

  //import ietf-netconf-acm {
  //  prefix nacm;
  //  reference
  //    "RFC 6536: Network Configuration Protocol (NETCONF)
  //      Access Control Model";
  //}

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

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
```

```
"RFC 7407: A YANG Data Model for SNMP Configuration";
}

import ietf-tls-server {
  prefix ts;
  revision-date 2016-11-02; // stable grouping definitions
  reference
    "RFC ZZZZ: TLS Client and Server Models";
}

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 XXXX; see
  the RFC itself for full legal notices.";

revision "2016-11-02" {
  description
    "Initial version";
  reference
```

```
    "RFC XXXX: RESTCONF Client and Server Models";
}

// Features

feature listen {
    description
        "The 'listen' feature indicates that the RESTCONF server
        supports opening a port to accept RESTCONF client connections
        using at least one transport (e.g., TLS, etc.).";
}

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

feature call-home {
    description
        "The 'call-home' feature indicates that the RESTCONF server
        supports initiating RESTCONF call home connections to REETCONF
        clients using at least one transport (e.g., TLS, etc.).";
    reference
        "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-call-home {
    description
        "The 'tls-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
container restconf-server {
  description
    "Top-level container for RESTCONF server configuration.";

  container listen {
    if-feature listen;
    description
      "Configures listen behavior";
    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 {
      if-feature tls-listen;
      container tls {
        description
          "TLS-specific listening configuration for inbound
           connections.";
        uses ts:listening-tls-server-grouping {
          refine port {
            default 443;
          }
          augment "client-auth" {
            description
              "Augments in the cert-to-name structure.";
            uses cert-maps-grouping;
          }
        }
      }
    }
  }
}
```



```

    }
  }
}

container call-home {
  if-feature 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 {
        if-feature tls-call-home;
        container tls {
          description
            "Specifies TLS-specific call-home transport
            configuration.";
          uses endpoints-container {
            refine endpoints/endpoint/port {
              default 4336;
            }
          }
          uses ts:non-listening-tls-server-grouping {
            augment "client-auth" {
              description
                "Augments in the cert-to-name structure.";
              uses cert-maps-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 maximum 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 client must close the
           connection when it's ready to release it. Once the
           connection has been closed, the server will restart
           its timer until the 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 to
             the RESTCONF client before this time if desired
             (e.g., to deliver a notification).";
        }
      }
    }
  }
}

container reconnect-strategy {
  description
    "The reconnection strategy directs how a RESTCONF server
     reconnects to a RESTCONF client after after discovering
     its connection to the client has dropped, 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).";
}
}
}
}
}

grouping cert-maps-grouping {
    description
        "A grouping that defines a container around the
        cert-to-name structure defined in RFC 7407.";
    container cert-maps {
        uses x509c2n:cert-to-name;
        description
            "The cert-maps container is used by a TLS-based RESTCONF
            server to map the RESTCONF client's presented X.509
            certificate to a RESTCONF username.  If no matching and
            valid cert-to-name list entry can be found, then the
            RESTCONF server MUST close the connection, and MUST NOT
            accept RESTCONF messages over it.";
        reference
            "RFC XXXX: The RESTCONF Protocol";
    }
}

```

```
grouping endpoints-container {
  description
    "This grouping is used by tls container 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 RESTCONF 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 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;
        description
          "The IP port for this endpoint. The RESTCONF server will
          use the IANA-assigned well-known port if no value is
          specified.";
      }
    }
  }
}
```

<CODE ENDS>

4. Security Considerations

5. IANA Considerations

5.1. The IETF XML Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

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

name: ietf-restconf-client
namespace: urn:ietf:params:xml:ns:yang:ietf-restconf-client
prefix: ncc
reference: RFC XXXX

name: ietf-restconf-server
namespace: urn:ietf:params:xml:ns:yang:ietf-restconf-server
prefix: ncs
reference: RFC XXXX

6. 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, Sean Turner, 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.

7. References

7.1. Normative References

- [draft-ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-ietf-netconf-keystore>>.
- [draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-13 (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-ietf-netconf-restconf>>.
- [draft-ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-00 (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-ietf-netconf-tls-client-server>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <<http://www.rfc-editor.org/info/rfc7407>>.

7.2. Informative References

- [draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), 2015, <<https://datatracker.ietf.org/html/draft-ietf-netconf-call-home-17>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Added in new features 'listen' and 'call-home' so future transports can be augmented in.

Appendix B. Open Issues

Please see: <https://github.com/netconf-wg/restconf-client-server/issues>.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: April 2, 2017

E. Voit
A. Clemm
A. Gonzalez Prieto
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Bierman
YumaWorks
September 29, 2016

Restconf and HTTP Transport for Event Notifications
draft-ietf-netconf-restconf-notif-01

Abstract

This document defines Restconf, HTTP2, and HTTP1.1 bindings for the transport Subscription requests and corresponding push updates. Being subscribed may be both Event Notifications and YANG Datastores.

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 April 2, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. Solution	4
3.1. Mechanisms for Subscription Establishment and Maintenance	4
3.2. Dynamic YANG Subscription with RESTCONF control	5
3.3. Subscription Multiplexing	8
3.4. Encoded Subscription and Event Notification Examples	9
3.5. Stream Discovery	14
4. Security Considerations	14
5. Acknowledgments	15
6. References	15
6.1. Normative References	15
6.2. Informative References	16
Appendix A. Proxy YANG Subscription when the Subscriber and Receiver are different	17
Appendix B. End-to-End Deployment Guidance	17
B.1. Call Home	17
B.2. TLS Heartbeat	18
Appendix C. Issues being worked and resolved	18
C.1. Unresolved Issues	18
C.2. Agreement in principal	18
C.3. Resolved Issues	18
Appendix D. Changes between revisions	19
Authors' Addresses	19

1. Introduction

Mechanisms to support Event subscription and push are defined in [rfc5277bis]. Enhancements to [rfc5277bis] which enable YANG Datastore subscription and push are defined in [yang-push]. This document provides a transport specification for these protocols over Restconf and HTTP. Driving these requirements is [RFC7923].

The streaming of Subscription Event Notifications has synergies with HTTP2 streams. Benefits which can be realized when transporting events directly HTTP2 [RFC7540] include:

- o Elimination of head-of-line blocking
- o Weighting and proportional dequeuing of Events from different subscriptions

- o Explicit precedence in subscriptions so that events from one subscription must be sent before another dequeues

2. Terminology

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

Configured Subscription: a Subscription installed via a configuration interface which persists across reboots.

Dynamic Subscription: a Subscription negotiated between Subscriber and Publisher via create, establish, modify, and delete RPC signaling messages.

Event: an occurrence of something that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, status of a flow, or an external input to the system.)

Event Notification: a set of information intended for a Receiver indicating that one or more Event(s) have occurred. Details of the Event(s) may be included within.

Event Stream: a continuous, ordered set of Events grouped under an explicit criteria.

Notification: the communication of an occurrence, perhaps triggered by the occurrence of an Event.

Publisher: an entity responsible for streaming Event Notifications per the terms of a Subscription.

Receiver: a target to which a Publisher pushes Event Notifications. For Dynamic Subscriptions, the Receiver and Subscriber will often be the same entity.

Subscriber: an entity able to request and negotiate a contract for the receipt of Event Notifications from a Publisher

Subscription: a contract between a Subscriber and a Publisher stipulating which information the Receiver wishes to have pushed from the Publisher without the need for further solicitation.

3. Solution

Event subscription is defined in [rfc5277bis], YANG Datastore subscription is defined in [yang-push]. This section specifies transport mechanisms applicable to both.

3.1. Mechanisms for Subscription Establishment and Maintenance

There are three models for Subscription establishment and maintenance:

1. Dynamic Subscription: Here the Subscriber and Receiver are the same. A Subscription ends with a subscription-terminated notification, or by a loss of transport connectivity.
2. Configured Subscription: Receiver(s) are specified on Publisher in startup and running config. Subscription is not terminated except via an operations interface. (Subscriptions may be Suspended, with no Event Notifications sent however.)
3. Proxy Subscription: Subscriber and Receiver are different. Subscription ends when a Subscription End-time is reached, or the Publisher process is restarted. A key difference between this and configured subscriptions (#2) is that configuration requests are made to RPCs which might evaluate run-time conditions much like in (#1). Typically direct configuration via (#2) will not go through the same sort of capacity and validation checks seen in (#1).

The first two models are described in this section. The third is described in Appendix A. This third model will be moved into the body of this specification should the IETF community desire. In theory, all three models may be intermixed in a single deployment.

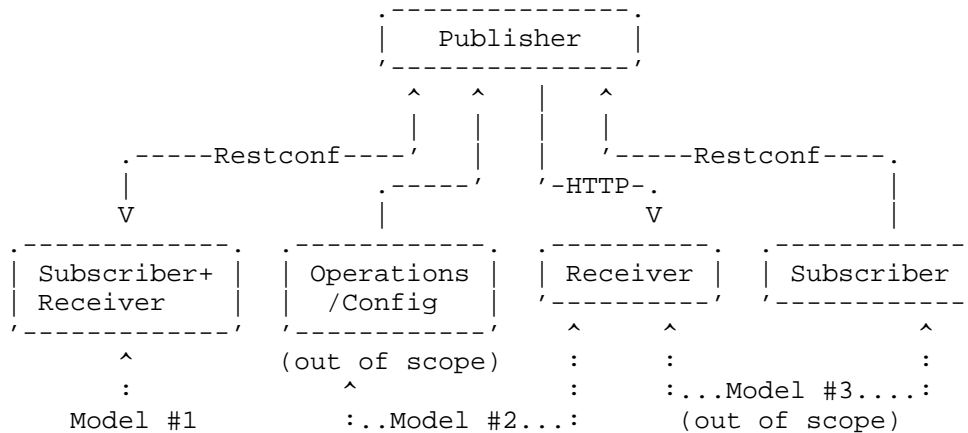


Figure 1: Subscription Models

3.2. Dynamic YANG Subscription with RESTCONF control

Dynamic Subscriptions for both [rfc5277bis] and its [yang-push] augmentations are configured and managed via signaling messages transported over [restconf]. These interactions will be accomplished via a Restconf POST into RPCs located on the Publisher. HTTP responses codes will indicate the results of the interaction with the Publisher. An HTTP status code of 200 is the proper response to a successful <establish-subscription> RPC call. The successful <establish-subscription> will result in a HTTP message with returned subscription URI on a logically separate mechanism than was used for the original Restconf POST. This mechanism would be via a parallel TCP connection in the case of HTTP 1.x, or in the case of HTTP2 via a separate HTTP stream within the HTTP connection. When a being returned by the Publisher, failure will be indicated by error codes transported in payload, as well as the return of negotiation parameters.

Once established, streaming Event Notifications are then delivered via SSE for HTTP1.1 and via HTTP Data for HTTP2.

3.2.1. Call Flow for HTTP2

Requests to [yang-push] augmented RPCs are sent on one or more HTTP2 streams indicated by (a) in Figure 2. Event Notifications related to a single subscription are pushed on a unique logical channel (b). In the case below, a newly established subscription has its events pushed over HTTP2 stream (7).

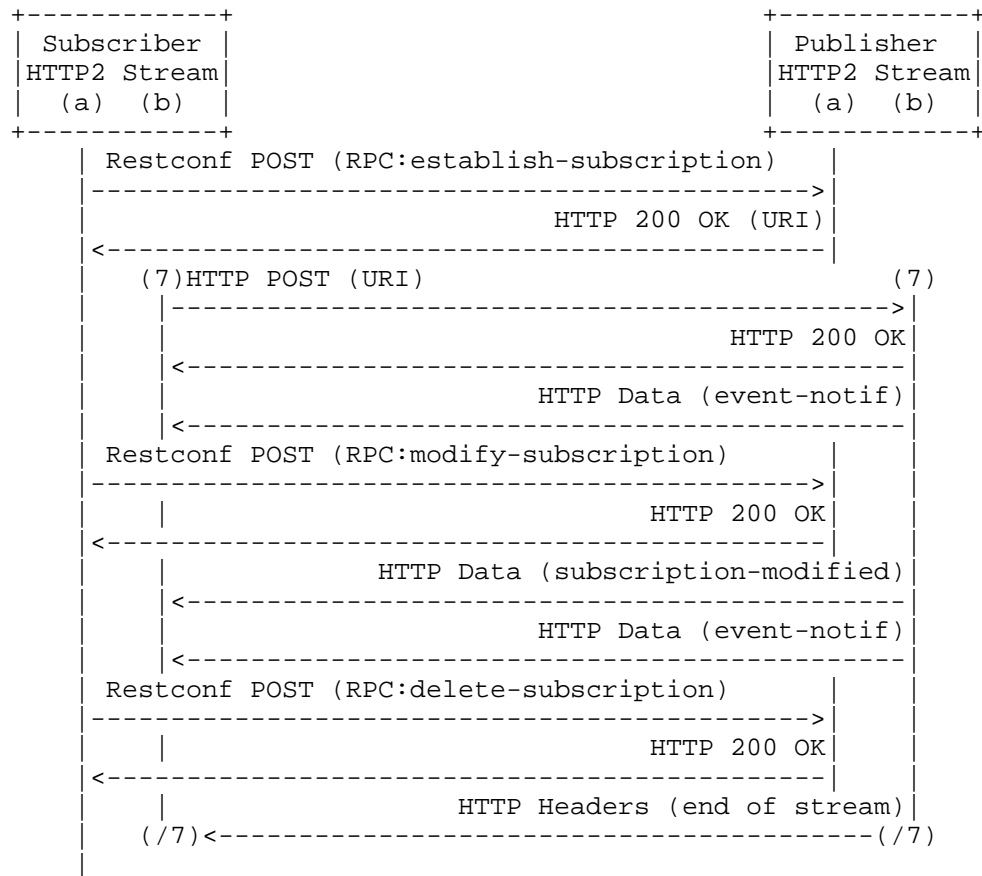


Figure 2: Dynamic with HTTP2

3.2.2. Call flow for HTTP1.1

Requests to [yang-push] RPCs are sent on the TCP connection indicated by (a). Event Notifications are pushed on a separate connection (b). This connection (b) will be used for all Event Notifications across all subscriptions.

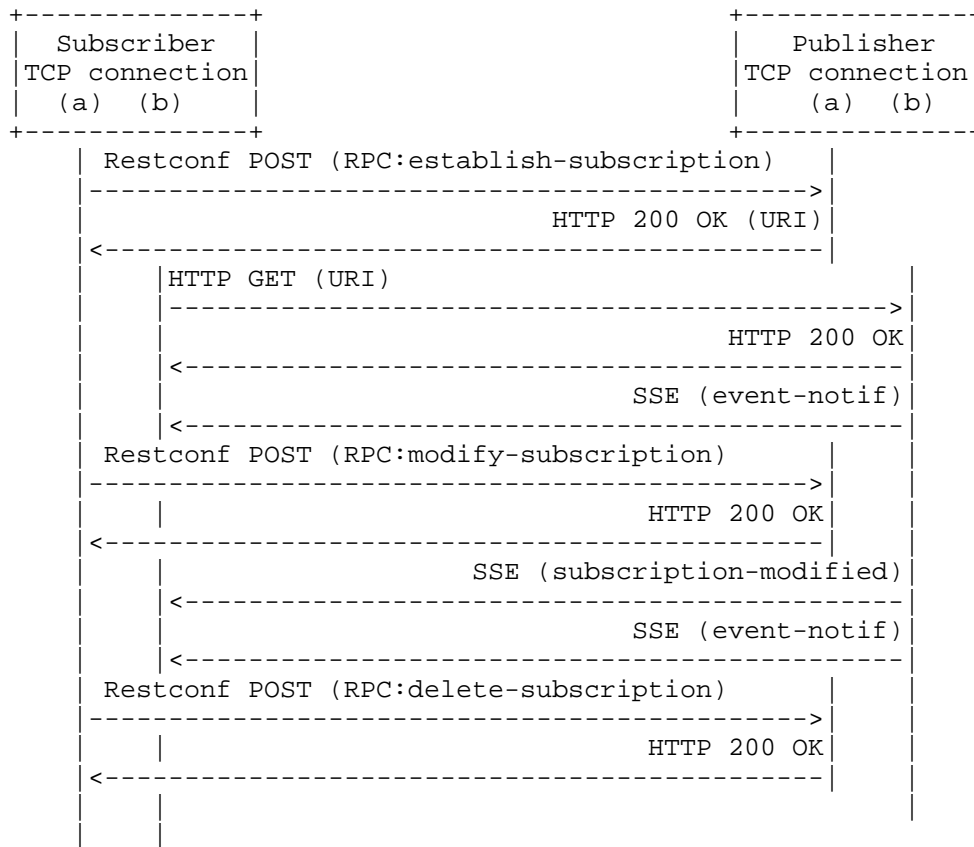


Figure 3: Dynamic with HTTP1.1

3.2.3. Configured Subscription over HTTP2

With a Configured Subscription, all information needed to establish a secure relationship with that Receiver is available on the Publisher. With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the Event Notifications to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that such Event Notifications be pushed with any dependency on Restconf. Nor is there value which Restconf provides on top of HTTP. Therefore in place of Restconf, a TLS secured HTTP2 Client connection must be established with an HTTP2 Server located on the Receiver. Event Notifications will then be sent as part of an extended HTTP POST to the Receiver.

POST messages will be addressed to HTTP augmentation code on the Receiver capable of accepting and responding to Event Notifications. The first POST message must be a subscription-started notification. Push update notifications must not be sent until the receipt of an HTTP 200 OK for this initial notification. The 200 OK will indicate that the Receiver is ready for Event Notifications. At this point a Subscription must be allocated its own HTTP2 stream. Figure 4 depicts this message flow.

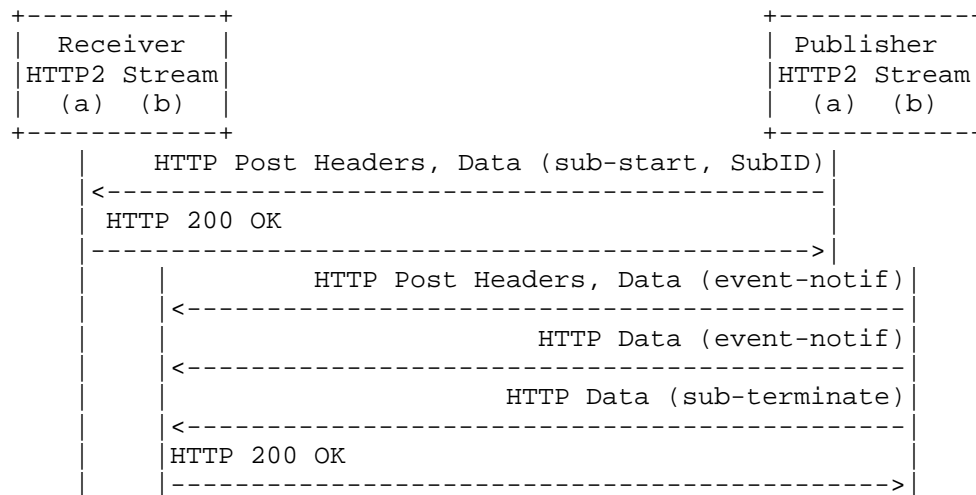


Figure 4: Configured over HTTP2

As the HTTP2 transport is available to the Receiver, the Publisher should:

- o take any subscription-priority and copy it into the HTTP2 stream priority, and
- o take a subscription-dependency if it has been provided and map the HTTP2 stream for the parent subscription into the HTTP2 stream dependency.

3.3. Subscription Multiplexing

It is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- o replay of pushed updates

- o temporary loss of transport connectivity, with update buffering and different dequeuing priorities per Subscription
- o population, marshalling and bundling of independent Subscription Updates, and

Therefore each Event Notification will include a millisecond level timestamp to ensure that a Receiver understands the time when a that update was generated. Use of this timestamp can give an indication of the state of objects at a Publisher when state-entangled information is received across different subscriptions. The use of the latest Event Notification timestamp for a particular object update can introduce errors. So when state-entangled updates have inconsistent object values and temporally close timestamps, a Receiver might consider performing a GET to validate the current state of a Publisher.

3.4. Encoded Subscription and Event Notification Examples

Transported updates will contain context data for one or more Event Notifications. Each transported Event Notification will contain several parameters:

3.4.1. Restconf Subscription and Events over HTTP1.1

Subscribers can dynamically learn whether a RESTCONF server supports various types of Event or Yang datastore subscription capabilities. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream. Some examples building upon the Call flow for HTTP1.1 from Section 3.2.2 are:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/  
    streams/stream=yang-push HTTP/1.1  
Host: example.com  
Accept: application/yang.data+xml
```

If the server supports it, it may respond

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
<stream xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  <name>yang-push</name>
  <description>Yang push stream</description>
  <access>
    <encoding>xml</encoding>
    <location>https://example.com/streams/yang-push-xml
  </access>
  <access>
    <encoding>json</encoding>
    <location>https://example.com/streams/yang-push-json
  </access>
</stream>
```

If the server does not support any form of subscription, it may respond

```
HTTP/1.1 404 Not Found
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
```

Subscribers can determine the URL to receive updates by sending an HTTP GET as a request for the "location" leaf with the stream list entry. The stream to use for may be selected from the Event Stream list provided in the capabilities exchange. Note that different encodings are supporting using different Event Stream locations. For example, the Subscriber might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
    streams/stream=yang-push/access=xml/location HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

The Publisher might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
  <location
    xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
    https://example.com/streams/yang-push-xml
  </location>
```

To subscribe and start receiving updates, the subscriber can then send an HTTP GET request for the URL returned by the Publisher in the request above. The accept header must be "text/event-stream". The

Publisher uses the Server Sent Events [W3C-20150203] transport strategy to push filtered Event Notifications from the Event stream.

The Publisher MUST support individual parameters within the POST request body for all the parameters of a subscription. The only exception is the encoding, which is embedded in the URI. An example of this is:

```
// subtree filter = /foo
// periodic updates, every 5 seconds
POST /restconf/operations/ietf-event-notifications:
  establish-subscription HTTP/1.1
  Host: example.com
  Content-Type: application/yang-data+json

  {
    "ietf-event-notifications:input" : {
      ?stream?: ?push-data"
      ?period" : 5,
      "xpath-filter" : ?/ex:foo[starts-with(?bar?.?some']"
    }
  }
```

Should the publisher not support the requested subscription, it may reply:

```

HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+xml
  <errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
    <error>
      <error-type>application</error-type>
      <error-tag>operation-not-supported</error-tag>
      <error-severity>error</error-severity>
      <error-message>Xpath filters not supported</error-message>
      <error-info>
        <supported-subscription xmlns="urn:ietf:params:xml:ns:
          netconf:datastore-push:1.0">
          <subtree-filter/>
        </supported-subscription>
      </error-info>
    </error>
  </errors>

```

with an equivalent JSON encoding representation of:

```

HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+json
  {
    "ietf-restconf:errors": {
      "error": {
        "error-type": "protocol",
        "error-tag": "operation-not-supported",
        "error-message": "Xpath filters not supported."
        "error-info": {
          "datastore-push:supported-subscription": {
            "subtree-filter": [null]
          }
        }
      }
    }
  }

```

The following is an example of a pushed Event Notification data for the Subscription above. It contains a subtree with root foo that contains a leaf called bar:

XML encoding representation:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <subscription-id xmlns="urn:ietf:params:xml:ns:restconf:
    datastore-push:1.0">
    my-sub
  </subscription-id>
  <eventTime>2015-03-09T19:14:56.23Z</eventTime>
  <datastore-contents xmlns="urn:ietf:params:xml:ns:restconf:
    datastore-push:1.0">
    <foo xmlns="http://example.com/yang-push/1.0">
      <bar>some_string</bar>
    </foo>
  </datastore-contents>
</notification>
```

Or with the equivalent YANG over JSON encoding representation as defined in [RFC7951]:

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.23Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a Subscription, the subscriber issues another POST request on the provided URI using the same subscription-id as in the original request. For example, to modify the update period to 10 seconds, the subscriber may send:

```
POST /restconf/operations/ietf-event-notifications:
modify-subscription HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json

{
  "ietf-event-notifications:input" : {
    ?subscription-id?: 100,
    ?period" : 10
  }
}
```

To delete a Subscription, the Subscriber issues a DELETE request on the provided URI using the same subscription-id as in the original request

3.4.2. Event Notification over HTTP2

The basic encoding will look as below. It will consists of a JSON representation wrapped in an HTTP2 header.

HyperText Transfer Protocol 2

Stream: HEADERS, Stream ID: 5

Header: :method: POST

Stream: HEADERS, Stream ID: 5

```
{
  "ietf-yangpush:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.23Z",
    "datastore-push:datastore-contents": {
      "foo": { "bar": "some_string" }
    }
  }
}
```

3.5. Stream Discovery

Relevant for Dynamic Subscriptions, this will be accomplished as specified in [restconf] section 6.2. The namespace chosen will be the same as how stream names are acquired for NETCONF, and so that backwards compatibility can be maintained without replicating this information.

As per [restconf] section 6.3, RESTCONF clients can determine the URL for the subscription resource (to receive notifications) by sending an HTTP GET request for the "location" leaf with the stream list entry.

4. Security Considerations

Subscriptions could be used to intentionally or accidentally overload the resources of a Publisher. For this reason, it is important that the Publisher has the ability to prioritize the establishment and push of Event Notifications where there might be resource exhaust potential. In addition, a server needs to be able to suspend existing Subscriptions when needed. When this occurs, the subscription status must be updated accordingly and the Receivers notified.

A Subscription could be used to attempt retrieve information for which a Receiver has no authorized access. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a Receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model [RFC6536] applies even though the transport is not NETCONF.

One or more Publishers of Configured Subscriptions could be used to overwhelm a Receiver which doesn't even support Subscriptions. There are two protections here. First Event Notifications for Configured Subscriptions MUST only be transmittable over Encrypted transports. Clients which do not want pushed Event Notifications need only terminate or refuse any transport sessions from the Publisher. Second, the HTTP transport augmentation on the Receiver must send an HTTP 200 OK to a subscription started notification before the Publisher starts streaming any events.

One or more Publishers could overwhelm a Receiver which is unable to control or handle the volume of Event Notifications received. In deployments where this might be a concern, HTTP2 transport such as HTTP2) should be selected.

5. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, Michael Scharf, and Guangying Zheng.

6. References

6.1. Normative References

- [restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", March 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf/>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <<http://www.rfc-editor.org/info/rfc6520>>.

- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<http://www.rfc-editor.org/info/rfc7923>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.

6.2. Informative References

- [call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", December 2015, <<https://tools.ietf.org/html/draft-ietf-netconf-call-home-17>>.
- [rfc5277bis]
Gonzalez Prieto, A., Clemm, A., Voit, E., Prasad Tripathy, A., and E. Nilsen-Nygaard, "NETCONF Event Notifications", September 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-rfc5277bis/>>.
- [W3C-20150203]
"Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", February 2015, <<https://www.w3.org/TR/2015/REC-eventsource-20150203/>>.
- [yang-push]
Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", June 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

Appendix A. Proxy YANG Subscription when the Subscriber and Receiver are different

The properties of Dynamic and Configured Subscriptions can be combined to enable deployment models where the Subscriber and Receiver are different. Such separation can be useful with some combination of:

- o An operator does not want the subscription to be dependent on the maintenance of transport level keep-alives. (Transport independence provides different scalability characteristics.)
- o There is not a transport session binding, and a transient Subscription needs to survive in an environment where there is unreliable connectivity with the Receiver and/or Subscriber.
- o An operator wants the Publisher to include highly restrictive capacity management and Subscription security mechanisms outside of domain of existing operational or programmatic interfaces.

To build a Proxy Subscription, first the necessary information must be signaled as part of the <establish-subscription>. Using this set of Subscriber provided information; the same process described within section 3 will be followed. There is one exception. Only when an HTTP status code of 200 comes back from the receiver, will it inform the Subscriber of Subscription establishment success via its Restconf connection.

After a successful establishment, if the Subscriber wishes to track the state of Receiver subscriptions, it may choose to place a separate on-change Subscription into the "Subscriptions" subtree of the YANG Datastore on the Publisher.

Appendix B. End-to-End Deployment Guidance

Several technologies are expected to be seen within a deployment to achieve security and ease-of-use requirements. These are not necessary for an implementation of this specification, but will be useful to consider when considering the operational context.

B.1. Call Home

Pub/Sub implementations should have the ability to transparently incorporate [call-home] so that secure TLS connections can originate from the desired device.

B.2. TLS Heartbeat

HTTP sessions might not quickly allow a Subscriber to recognize when the communication path has been lost from the Publisher. To recognize this, it is possible for a Receiver to establish a TLS heartbeat [RFC6520]. In the case where a TLS heartbeat is included, it should be sent just from Receiver to Publisher. Loss of the heartbeat should result in any Subscription related TCP sessions between those endpoints being torn down. The subscription can then attempt to re-establish.

Appendix C. Issues being worked and resolved

(To be removed by RFC editor prior to publication)

C.1. Unresolved Issues

RT3 - Do we include 3rd party signaled subscriptions within models that need to be supported generically, or for a particular type of transport.

RT10 - Right now the examples show a YANG timestamp at the hundredths of a second level. But the yang-push draft is at seconds. And the requirements show at least milliseconds (if not more).

C.2. Agreement in principal

RT4 - Need to add into document examples of 5277bis Event streams. Document only includes yang-push examples at this point.

RT6 - We need to define encodings of rfc5277bis notifications.

C.3. Resolved Issues

RT1 - Integration specifics for Restconf capability discovery on different types of Streams

RT2 - In what way to we position Event notifications model in this document vs. current solution in Restconf.

RT5 - Doesn't make sense to use Restconf for Configured subscriptions. HTTP will be used.

RT7 - HTTP native option doesn't currently use SSE. But we should evaluate moving to that as possible. It will make development integration easier and more consistent.

RT8 - Once SSE starts, there will be no more Restconf interpretation of further signaling upon the connection. It is unclear how this can be made to work with modify and delete subscription. If it cannot, a method of sending events without SSE will be needed, although this would diverge from the existing Restconf mechanisms

RT9 - For static subscriptions, perhaps we can use Restconf call home to originate an SSE connection. This assume RT8 & RT2 can be resolved with SSE.

Appendix D. Changes between revisions

(To be removed by RFC editor prior to publication)

v00 - v01

- o Removed the ability for more than one subscription to go to a single HTTP2 stream.
- o Updated call flows. Extensively.
- o SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions
- o HTTP is not used to determine that a Receiver has gone silent and is not Receiving Event Notifications
- o Many clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Cisco Systems

Email: alex@clemm.org

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2017

A. Clemm
Sympotech
A. Gonzalez Prieto
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
S. Chisholm
Ciena
H. Trevino
Cisco Systems
October 27, 2016

Subscribing to Event Notifications
draft-ietf-netconf-rfc5277bis-01

Abstract

This document defines capabilities and operations for subscribing to content and providing asynchronous notification message delivery on that content. Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF and RESTCONF. The capabilities and operations defined in this document when using in conjunction with draft-ietf-netconf-netconf-event-notifications are intended to replace RFC 5277.

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 April 30, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Motivation	3
1.2. Terminology	4
1.3. Solution Overview	5
2. Solution	6
2.1. Event Streams	6
2.2. Event Stream Discovery	7
2.3. Filters	7
2.4. Subscription State Model at the Publisher	7
3. Data Model Trees for Event Notifications	8
4. Dynamic Subscriptions	12
4.1. Establishing a Subscription	12
4.2. Modifying a Subscription	13
4.3. Deleting a Subscription	13
5. Configured Subscriptions	14
5.1. Establishing a Configured Subscription	14
5.2. Modifying a Configured Subscription	16
5.3. Deleting a Configured Subscription	16
6. Event (Data Plane) Notifications	17
7. Control Plane Notifications	18
7.1. replayComplete	19
7.2. notificationComplete	19
7.3. subscription-started	19
7.4. subscription-modified	19
7.5. subscription-terminated	19
7.6. subscription-suspended	20
7.7. subscription-resumed	20
8. Data Model for Event Notifications	20
9. Backwards Compatibility	40
10. Security Considerations	41
11. Acknowledgments	42

12. References	42
12.1. Normative References	42
12.2. Informative References	42
Appendix A. Issues that are currently being worked and resolved	43
A.1. Unresolved and yet-to-be addressed issues	43
A.2. Agreement in principal	43
A.3. Resolved Issues	44
Appendix B. Changes between revisions	44
Authors' Addresses	45

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service in a protocol-agnostic manner. This document defines capabilities and operations for providing asynchronous message notification delivery for notifications including those necessary to establish, monitor, and support subscriptions to notification delivery.

Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF [RFC6241] (defined in [I-D.ietf-netconf-netconf-event-notif]) and Restconf [I-D.ietf-netconf-restconf] (defined in [I-D.ietf-netconf-restconf-notif]). The capabilities and operations defined in this document are intended to replace RFC 5277, along with their mapping onto NETCONF transport.

1.1. Motivation

The motivation for this work is to enable the sending of transport agnostic asynchronous notification messages driven by a YANG Subscription that are consistent with the data model (content) and security model. Predating this work was used within a NETCONF implementation. [RFC5277] which defined a limited defines a notification mechanism for for NETCONF. However, there are various [RFC5277] has limitations:, many of which have been exposed in [RFC7923].

The scope of the work aims at meeting the operational needs of network subscriptions:

- o Ability to dynamically or statically subscribe to event notifications available on a publisher.
- o Ability to negotiate acceptable dynamic subscription parameters.
- o Ability to filter the subset of notifications to be pushed with stream-specific semantics.

- o Ability for the notification payload to be interpreted independently of the transport protocol. (In other words, the encoded notification fully describes itself.)
- o Mechanism to communicate the notifications.
- o Ability to replay locally logged notifications.

1.2. Terminology

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

Configured subscription: A subscription installed via a configuration interface which persists across reboots.

Dynamic subscription: A subscription agreed between subscriber and publisher via create, establish, modify, and delete RPC control plane signaling messages.

Event: An occurrence of something that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.)

Event notification: A set of information intended for a Receiver indicating that one or more Event(s) have occurred. Details of the Event(s) may be included within the Notification.

Filter: Evaluation criteria, which may be applied against a targeted set of objects/events in a subscription. Information traverses the filter only if specified filter criteria are met.

NACM: NETCONF Access Control Model.

OAM: Operations, Administration, Maintenance.

Publisher: An entity responsible for streaming Event Notifications per the terms of a Subscriptions

Receiver: A target to which a publisher pushes event notifications. For dynamic subscriptions, the receiver and subscriber will often be the same entity.

RPC: Remote Procedure Call.

Stream (also referred to as "event stream"): A continuous ordered set of events grouped under an explicit criteria.

Subscriber: An entity able to request and negotiate a contract for the receipt of event notifications from a publisher.

Subscription: A contract with a publisher, stipulating which information receiver(s) wishes to have pushed from the publisher without the need for further solicitation.

1.3. Solution Overview

This document describes mechanisms for subscribing and receiving event notifications from an event server publisher. This document builds on top of the capabilities defined in [RFC5277], extending them, and generalizing them to be protocol-agnostic.

Some enhancements over RFC 5277 include the ability to have multiple subscriptions on a single transport session, to terminate a single subscriptions without terminating the transport session, and to modify existing subscriptions.

These enhancements do not affect existing RFC 5277 subscribers that do not support these particular subscription requirements.

The solution supports subscribing to event notifications using two mechanisms:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via RPC. A subscriber initiates a negotiation by issuing a subscription request. If the publisher wants to serve this request, it will accept it, and then start pushing event notifications as negotiated. If the publisher does not wish to serve it as requested, it may respond with subscription parameters which it would have accepted.
2. Configured subscriptions, which is an optional mechanism that enables managing subscriptions via a configuration interface so that a publisher can send event notifications to configured receiver(s).

Some key characteristics of configured and dynamic subscriptions include:

- o The lifetime of a dynamic subscription is limited by the lifetime of the subscriber session used to establish it. Typically loss of the transport session tears down any dependent dynamic subscriptions.
- o The lifetime of a configured subscription is driven by configuration being present on the running configuration. This

implies configured subscriptions persist across reboots, and persists even when transport is unavailable. This also means configured subscriptions do not support negotiation.

- o Subscriptions can be modified or terminated at any point of their lifetime. configured subscriptions can be modified by any configuration client with write rights on the configuration of the subscription.

Note that there is no mixing-and-matching of dynamic and configured subscriptions. Specifically, a configured subscription cannot be modified or deleted using RPC. Similarly, a subscription created via RPC cannot be modified through configuration operations.

The publisher may decide to terminate a dynamic subscription at any time. Similarly, it may decide to temporarily suspend the sending of event notifications for either configured or dynamic subscriptions. Such termination or suspension may be driven by the publisher running out of resources to serve the subscription, or by internal errors on the publisher.

2. Solution

2.1. Event Streams

An event stream is a set of events available for subscription from a publisher. It is out of the scope of this document to identify a) how streams are defined, b) how events are defined/generated, and c) how events are assigned to streams.

That said, some event streams will be standardized whereas others may be vendor specific. One standardized event stream is the "NETCONF" notification event stream. The NETCONF event stream contains all NETCONF XML event notifications supported by the publisher, except for those belonging only to streams that explicitly indicate that they must be excluded from the NETCONF stream, such as notifications that serve OAM and signaling purposes.

The following is a high-level description of the flow of a notification. Note that it does not mandate and/or preclude an implementation. As events are raised, they are assigned to streams. An event may be assigned to multiple streams. The event is distributed to subscribers and receivers based on the current subscriptions and access control. Access control is needed because if any receiver of that subscription does not have permission to receive an event, then it never makes it into a notification, and processing of the event is completed for that subscription.

2.2. Event Stream Discovery

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. A client can retrieve this list like any other YANG-defined data, for example using the <get> operation when using NETCONF.

2.3. Filters

a publisher implementation SHOULD support the ability to perform filtering of notification records per RFC 5277. (TODO: since 5277 is to be obsoleted, we should describe the filter here.)

2.4. Subscription State Model at the Publisher

Below is the state machine of a subscription for the publisher. It is important to note that a subscription doesn't exist at the publisher until it is accepted and made active. The mere request by a subscriber to establish a subscription is insufficient for that asserted subscription to be externally visible via this state machine.

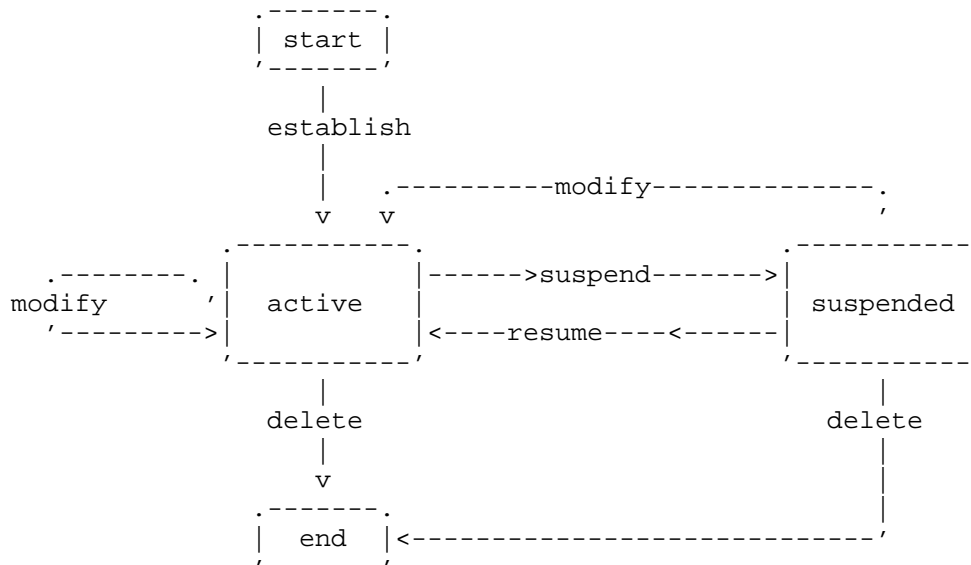


Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

- o Successful <establish-subscription> or <modify-subscription> requests put the subscription into an active state.
- o Failed <modify-subscription> requests will leave the subscription in its previous state, with no visible change to any streaming updates.
- o A <delete-subscription> request will delete the entire subscription.

3. Data Model Trees for Event Notifications

The YANG data model for event notifications is depicted in this section.

```

module: ietf-event-notifications
  +--ro streams
  |   +--ro stream*   stream
  +--rw filters
  |   +--rw filter* [filter-id]
  |   |   +--rw filter-id   filter-id
  |   |   +--rw (filter-type)?
  |   |   |   +---:(rfc5277)
  |   |   |   +--rw filter?
  |   +--rw subscription-config {configured-subscriptions}?
  |   |   +--rw subscription* [subscription-id]
  |   |   |   +--rw subscription-id   subscription-id
  |   |   |   +--rw stream?           stream
  |   |   |   +--rw encoding?         encoding
  |   |   |   +--rw (filter-type)?
  |   |   |   |   +---:(rfc5277)
  |   |   |   |   |   +--rw filter?
  |   |   |   |   |   +---:(by-reference)
  |   |   |   |   |   |   +--rw filter-ref?           filter-ref
  |   |   |   +--rw startTime?        yang:date-and-time
  |   |   |   +--rw stopTime?         yang:date-and-time
  |   |   +--rw receivers
  |   |   |   +--rw receiver* [address]
  |   |   |   |   +--rw address        inet:host
  |   |   |   |   +--rw port          inet:port-number
  |   |   |   |   +--rw protocol?     transport-protocol
  |   |   +--rw (push-source)?
  |   |   |   +---:(interface-originated)
  |   |   |   |   +--rw source-interface?  if:interface-ref
  |   |   |   +---:(address-originated)
  |   |   |   |   +--rw source-vrf?        uint32
  |   |   |   |   +--rw source-address    inet:ip-address-no-zone
  +--ro subscriptions

```

```

+--ro subscription* [subscription-id]
  +--ro subscription-id          subscription-id
  +--ro configured-subscription?
  |                             empty {configured-subscriptions}?
  +--ro subscription-status?     subscription-status
  +--ro stream?                  stream
  +--ro encoding?                encoding
  +--ro (filter-type)?
  |   +--:(rfc5277)
  |   |   +--ro filter?
  |   |   +--:(by-reference)
  |   |   +--ro filter-ref?          filter-ref
  +--ro startTime?              yang:date-and-time
  +--ro stopTime?              yang:date-and-time
  +--ro receivers
  |   +--ro receiver* [address]
  |   |   +--ro address          inet:host
  |   |   +--ro port            inet:port-number
  |   |   +--ro protocol?       transport-protocol
  +--ro (push-source)?
  |   +--:(interface-originated)
  |   |   +--ro source-interface?    if:interface-ref
  |   |   +--:(address-originated)
  |   |   +--ro source-vrf?          uint32
  |   |   +--ro source-address      inet:ip-address-no-zone

rpcs:
+---x establish-subscription
  +---w input
  |   +---w stream?          stream
  |   +---w encoding?       encoding
  |   +---w (filter-type)?
  |   |   +--:(rfc5277)
  |   |   |   +---w filter?
  |   |   |   +--:(by-reference)
  |   |   |   +---w filter-ref?    filter-ref
  |   +---w startTime?      yang:date-and-time
  |   +---w stopTime?      yang:date-and-time
  +--ro output
  |   +--ro subscription-result    subscription-result
  |   +--ro (result)?
  |   |   +--:(success)
  |   |   |   +--ro subscription-id    subscription-id
  |   |   +--:(no-success)
  |   |   +--ro stream?              stream
  |   |   +--ro encoding?            encoding
  |   |   +--ro (filter-type)?
  |   |   |   +--:(rfc5277)

```

```

|         | |  +--ro filter?
|         | |  +---:(by-reference)
|         | |  +--ro filter-ref?          filter-ref
|         | |  +--ro startTime?          yang:date-and-time
|         | |  +--ro stopTime?           yang:date-and-time
+---x create-subscription
|   +---w input
|     +---w stream?          stream
|     +---w encoding?       encoding
|     +---w (filter-type)?
|       | +---:(rfc5277)
|       | +---w filter?
|     +---w startTime?      yang:date-and-time
|     +---w stopTime?       yang:date-and-time
+---x modify-subscription
|   +---w input
|     +---w subscription-id?  subscription-id
|     +---w (filter-type)?
|       | +---:(rfc5277)
|       | | +---w filter?
|       | | +---:(by-reference)
|       | | +---w filter-ref?          filter-ref
|     +---w startTime?          yang:date-and-time
|     +---w stopTime?           yang:date-and-time
+---ro output
|   +---ro subscription-result  subscription-result
|   +---ro (result)?
|     +---:(success)
|       | +---ro subscription-id      subscription-id
|     +---:(no-success)
|       +---ro stream?              stream
|       +---ro encoding?            encoding
|       +---ro (filter-type)?
|         | +---:(rfc5277)
|         | | +---ro filter?
|         | | +---:(by-reference)
|         | | +---ro filter-ref?          filter-ref
|       +---ro startTime?          yang:date-and-time
|       +---ro stopTime?           yang:date-and-time
+---x delete-subscription
|   +---w input
|     | +---w subscription-id      subscription-id
+---ro output
|   +---ro subscription-result  subscription-result

notifications:
+---n replay-complete
|   +---ro subscription-id      subscription-id

```

```

+---n notification-complete
|   +---ro subscription-id    subscription-id
+---n subscription-started
|   +---ro subscription-id    subscription-id
|   +---ro stream?            stream
|   +---ro encoding?          encoding
|   +---ro (filter-type)?
|   |   +---:(rfc5277)
|   |   |   +---ro filter?
|   |   +---:(by-reference)
|   |   |   +---ro filter-ref?          filter-ref
|   +---ro startTime?          yang:date-and-time
|   +---ro stopTime?           yang:date-and-time
+---n subscription-suspended
|   +---ro subscription-id    subscription-id
|   +---ro reason?            subscription-susp-reason
+---n subscription-resumed
|   +---ro subscription-id    subscription-id
+---n subscription-modified
|   +---ro subscription-id    subscription-id
|   +---ro stream?            stream
|   +---ro encoding?          encoding
|   +---ro (filter-type)?
|   |   +---:(rfc5277)
|   |   |   +---ro filter?
|   |   +---:(by-reference)
|   |   |   +---ro filter-ref?          filter-ref
|   +---ro startTime?          yang:date-and-time
|   +---ro stopTime?           yang:date-and-time
+---n subscription-terminated
|   +---ro subscription-id    subscription-id
|   +---ro reason?            subscription-term-reason

```

The data model is structured as follows:

- o "Streams" contains a list of event streams that are supported by the publisher and that can be subscribed to.
- o "Filters" contains a configurable list of filters that can be applied to a subscription. This allows users to reference an existing filter definition as an alternative to defining a filter inline for each subscription.
- o "Subscription-config" contains the configuration of configured subscriptions. The parameters of each configured subscription are a superset of the parameters of a dynamic subscription and use the same groupings. In addition, the configured subscriptions must

also specify intended receivers and may specify the push source from which to send the stream of notification messages.

- o "Subscriptions" contains a list of all subscriptions on a publisher, both configured and dynamic. It can be used to retrieve information about the subscriptions which an publisher is serving.

The data model also contains a number of notifications that allow a publisher to signal information about a subscription. Finally, the data model contains a number of RPC definitions that are used to manage dynamic subscriptions.

4. Dynamic Subscriptions

Dynamic subscriptions are managed via RPC.

4.1. Establishing a Subscription

This operation includes and extends the "create-subscription" operation defined in RFC 5277. It allows a subscriber to request the creation of a subscription both via RPC and configuration operations. When invoking the RPC, establish-subscription permits negotiating the subscription terms, changing them dynamically.

The input parameters of the operation are those of create subscription plus:

- o filter-ref: filters that have been previously (and separately) configured can be referenced by a subscription. This mechanism enables the reuse of filters.
- o encoding: by default, updates are encoded using XML. Other encodings may be supported, such as JSON.

If the publisher cannot satisfy the request, it sends a negative <subscription-result> element.

If the subscriber has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. If the request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error what subscription parameters would have been accepted for the request when it was processed. However, there is no guarantee that subsequent requests with those parameters for this subscriber or others will be accepted. For instance, consider a subscription from [I-D.ietf-netconf-yang-push], which augments the establish-subscription with some additional parameters, including "period".

Subscription requests will fail if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

4.2. Modifying a Subscription

This operation permits modifying the terms of a dynamic subscription previously established. Subscriptions created by configuration cannot be modified. Dynamic subscriptions can be modified one or multiple times. If the publisher accepts the request, it immediately starts sending events based on the new terms, completely ignoring the previous ones. If the publisher rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of negative responses to modify-subscription requests are the subset of the establish subscription request parameters which are allowed to be dynamically modified.

Dynamic subscriptions established via RPC can only be modified (or deleted) via RPC using the same transport session used to establish that subscription.

Configured subscriptions cannot be modified (or deleted) using RPCs. Instead, configured subscriptions are modified (or deleted) as part of regular configuration operations. Publishers MUST reject any attempts to modify (or delete) configured subscriptions via RPC.

4.3. Deleting a Subscription

This operation permits canceling a subscription previously established. If the publisher accepts the request, it immediately stops sending events for the subscription. If the publisher rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever.

Subscriptions created via RPC can only be deleted via RPC using the same transport session used for subscription establishment. Configured subscriptions cannot be deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to delete configured subscriptions.

The only parameter to delete-subscription is the identifier of the subscription to delete.

If the publisher can satisfy the request, it sends an OK element.

If the publisher cannot satisfy the request, it sends an error-rpc element.

5. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface.

Configured subscriptions persist across reboots, and persist even when transport is unavailable. This also means configured subscriptions do not support negotiation.

Configured subscriptions can be modified by any configuration client with write permissions for the configuration of the subscription. Subscriptions can be modified or terminated at any point of their lifetime.

Supporting configured subscriptions is optional and advertised using the "configured-subscriptions" feature.

In addition to subscription parameters that apply to dynamic subscriptions, the following additional parameters apply to configured subscriptions:

- o One or more receiver IP addresses (and corresponding ports) intended as the destination for push updates for each subscription. In addition, the transport protocol for each destination may be defined.
- o Optional parameters to identify an egress interface or IP address / VRF where a subscription updates should be pushed from the publisher.

5.1. Establishing a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPC and configuration operations for subscription establishment. Firstly, configuration operations do not support negotiation while RPCs do. Secondly, while RPCs mandate that the subscriber establishing the subscription is the only receiver of the notifications, configuration operations permit specifying receivers independent of any tracked subscriber. Immediately after a subscription is successfully established, the publisher sends to its receivers a control-plane notification stating the subscription has been established (subscription-started).

Because there is no explicit association with an existing transport session, configured configuration operations require additional parameters to indicate the receivers of the notifications and

possibly the source of the notifications such as a specific egress interface.

For example at subscription establishment, a client may send:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
      <subscription>
        <subscription-id>
          1922
        </subscription-id>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.4
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 2: Establish configured subscription

if the request is accepted, the publisher would reply:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 3: Response to a successful configured subscription establishment

if the request is not accepted because the publisher cannot serve it, the publisher may reply:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>resource-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">
      Temporarily the publisher cannot serve this
      subscription due to the current workload.
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 4: Response to a failed configured subscription establishment

5.2. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level subtree subscription-config.

Immediately after a subscription is successfully modified, the publisher sends to the existing receivers a control-plane notification stating the subscription has been modified (i.e., subscription-modified).

If the modification involved adding and/or removing receivers, those modified receivers are sent control-plane notifications, indicating they have been added (i.e., subscription-started to a specific receiver) or removed (i.e., subscription-terminated to a specific receiver.)

5.3. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example, in RESTCONF:

```
DELETE /subscription-config/subscription=1922 HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Date: Sun, 24 Jul 2016 11:23:40 GMT
Server: example-server
```

Figure 5: Deleting a configured subscription

Immediately after a subscription is successfully deleted, the publisher sends to all receivers a control-plane notification stating the subscription has been terminated (subscription-terminated).

6. Event (Data Plane) Notifications

Once a subscription has been set up, the publisher streams (asynchronously) the event notifications per the terms of the subscription. We refer to these as data plane notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the session used to create or establish the subscription. For configured subscriptions, event notifications are sent over the specified connections.

An event notification is sent to the receiver(s) when an event of interest (i.e., meeting the specified filtering criteria) has occurred. An event notification is a complete and well-formed XML document. Note that `<notification>` is not a Remote Procedure Call (RPC) method but rather the top-level element identifying the one-way message as a notification. Note that event notifications never trigger responses.

The event notification always includes an `<eventTime>` element. It is the time the event was generated by the event source. This parameter is of type `dateTime` and compliant to [RFC3339]. Implementations must support time zones.

The event notifications must also include the subscription-id if the establish-subscription was used in its establishment, or if it was configured via an operational interface.

The event notification also contains notification-specific tagged content, if any.

The following is an example of an event notification from [RFC7950]:

```
notification link-failure {
  description "A link failure has been detected";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}
```

Figure 6: Definition of a data plane notification

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

Figure 7: Data plane notification

The equivalent using json encoding would be

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
    {
      "acme-system:link-failure": {
        "if-name": "so-1/2/3.0",
        "if-admin-status": "up",
        "if-oper-status": "down "
      }
    }
  </notification-contents-json>
</notification>
```

Figure 8: Data plane notification using JSON encoding

7. Control Plane Notifications

In addition to data plane notifications, a publisher may send control plane notifications to indicate to receivers that an event related to the subscription management has occurred.

Control plane notifications are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to the receiver of a subscription. The definition of control plane notifications is distinct from other notifications by making use of a YANG extension tagging them as control plane notification.

Control plane notifications include indications that a replay of notifications has been completed, that a subscription is done sending notifications because an end time has been reached, and that a subscription has started, been modified, been terminated, or been suspended. They are described in the following subsections.

7.1. replayComplete

This notification is originally defined in [RFC5277]. It is sent to indicate that all of the replay notifications have been sent. This notification must not be sent for any other reason.

In the case of a subscription without a stop time or a stop time which has not been reached, after the <replayComplete> notification has been sent, it can be expected that any notifications generated since the start of the subscription creation will be sent, followed by notifications in sequence as they arise naturally within the system.

7.2. notificationComplete

This notification is originally defined in [RFC5277]. It is sent to indicate that a subscription, which includes a stop time, has finished passing events.

7.3. subscription-started

This notification indicates that a configured subscription has started and data updates are beginning to be sent. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

7.4. subscription-modified

This notification indicates that a configured subscription has been modified successfully. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

7.5. subscription-terminated

This notification indicates that a subscription has been terminated by the publisher. The notification includes the reason for the termination. The publisher may decide to terminate a subscription when it is running out of resources for serving it, an internal error occurs, etc. Publisher-driven terminations are notified to all receivers. The management plane can also terminate configured subscriptions using configuration operations.

Subscribers can terminate via RPC subscriptions established via RPC. In such cases, no subscription-terminated notifications are sent.

7.6. subscription-suspended

This notification indicates that a publisher has suspended a subscription. The notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further data plane notifications will be sent until the subscription resumes. Suspensions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

7.7. subscription-resumed

This notification indicates that a previously suspended subscription has been resumed. Data plane notifications generated in the future will be sent after the subscription terms. Resumptions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

8. Data Model for Event Notifications

```
<CODE BEGINS> file "ietf-event-notifications@2016-10-27.yang"
module ietf-event-notifications {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-event-notifications";

  prefix notif-bis;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-interfaces {
    prefix if;
  }

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

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

     WG Chair: Mehmet Ersue
               <mailto:mehmet.ersue@nokia.com>;"
```

Editor: Alexander Clemm
<mailto:alex@sympotech.com>
Editor: Alberto Gonzalez Prieto
<mailto:albertgo@cisco.com>
Editor: Eric Voit
<mailto:evoit@cisco.com>
Editor: Einar Nilsen-Nygaard
<mailto:einarnn@cisco.com>
Editor: Ambika Prasad Tripathy
<mailto:ambtripa@cisco.com>
Editor: Sharon Chisholm
<mailto:schishol@ciena.com>
Editor: Hector Trevino
<mailto:htrevino@cisco.com>

description

"This module contains conceptual YANG specifications
for NETCONF Event Notifications.";

revision 2016-10-27 {

description

"Tweaks to remove two notifications,
RPC for create subscription refined with stream default,
new grouping to eliminate some dynamically modifiable
parameters in modify subscription RPC";

reference

"draft-ietf-netconf-rfc5277bis-01";

}

/*

* FEATURES

*/

feature json {

description

"This feature indicates that JSON encoding of notifications
is supported.";

}

feature configured-subscriptions {

description

```
        "This feature indicates that management plane configuration
        of subscription is supported.";
    }

    /*
    * EXTENSIONS
    */

    extension control-plane-notif {
        description
            "This statement applies only to notifications.
            It indicates that the notification is a control-plane
            notification (aka OAM notification). Therefore it does
            not participate in a regular event stream and does not
            need to be specifically subscribed to.";
    }

    /*
    * IDENTITIES
    */

    /* Identities for streams */
    identity stream {
        description
            "Base identity to represent a generic stream of event
            notifications.";
    }

    identity NETCONF {
        base stream;
        description
            "Default NETCONF event stream, containing events based on
            notifications defined as YANG modules that are supported
            by the system.";
    }

    /* Identities for subscription results */
    identity subscription-result {
        description
            "Base identity for RPC responses to requests surrounding
            management (e.g. creation, modification, deletion) of
            subscriptions.";
    }

    identity ok {
        base subscription-result;
        description
            "OK - RPC was successful and was performed as requested.";
    }
```

```
}

identity error {
  base subscription-result;
  description
    "RPC was not successful.
    Base identity for error return codes.";
}

identity error-no-such-subscription {
  base error;
  description
    "A subscription with the requested subscription ID
    does not exist.";
}

identity error-no-such-option {
  base error;
  description
    "A requested parameter setting is not supported.";
}

identity error-insufficient-resources {
  base error;
  description
    "The publisher has insufficient resources to support the
    subscription as requested.";
}

identity error-configured-subscription {
  base error;
  description
    "Cannot apply RPC to a configured subscription, i.e.
    to a subscription that was not established via RPC.";
}

identity error-other {
  base error;
  description
    "An unspecified error has occurred (catch all).";
}

/* Identities for subscription stream status */
identity subscription-stream-status {
  description
    "Base identity for the status of subscriptions and
    datastreams.";
}
```

```
identity active {
  base subscription-stream-status;
  description
    "Status is active and healthy.";
}

identity inactive {
  base subscription-stream-status;
  description
    "Status is inactive, for example outside the
    interval between start time and stop time.";
}

identity suspended {
  base subscription-stream-status;
  description
    "The status is suspended, meaning that the publisher
    is currently unable to provide the negotiated updates
    for the subscription.";
}

identity in-error {
  base subscription-stream-status;
  description
    "The status is in error or degraded, meaning that
    stream and/or subscription is currently unable to provide
    the negotiated notifications.";
}

/* Identities for subscription errors */
identity subscription-errors {
  description
    "Base identity for subscription error status.
    This identity is not to be confused with error return
    codes for RPCs";
}

identity internal-error {
  base subscription-errors;
  description
    "Subscription failures caused by server internal error.";
}

identity no-resources {
  base subscription-errors;
  description
    "Lack of resources, e.g. CPU, memory, bandwidth";
}
```

```
identity subscription-deleted {
  base subscription-errors;
  description
    "The subscription was terminated because the subscription
    was deleted.";
}

identity other {
  base subscription-errors;
  description
    "Fallback reason - any other reason";
}

/* Identities for encodings */
identity encodings {
  description
    "Base identity to represent data encodings";
}

identity encode-xml {
  base encodings;
  description
    "Encode data using XML";
}

identity encode-json {
  base encodings;
  description
    "Encode data using JSON";
}

/* Identities for transports */
identity transport {
  description
    "An identity that represents a transport protocol for
    event notifications";
}

identity netconf {
  base transport;
  description
    "Netconf notifications as a transport.";
}

/*
 * TYPEDEFS
 */
```

```
typedef subscription-id {
    type uint32;
    description
        "A type for subscription identifiers.";
}

typedef filter-id {
    type uint32;
    description
        "A type to identify filters which can be associated with a
        subscription.";
}

typedef subscription-result {
    type identityref {
        base subscription-result;
    }
    description
        "The result of a subscription operation";
}

typedef subscription-term-reason {
    type identityref {
        base subscription-errors;
    }
    description
        "Reason for a publisher to terminate a subscription.";
}

typedef subscription-susp-reason {
    type identityref {
        base subscription-errors;
    }
    description
        "Reason for a publisher to suspend a subscription.";
}

typedef encoding {
    type identityref {
        base encodings;
    }
    description
        "Specifies a data encoding, e.g. for a data subscription.";
}

typedef subscription-status {
    type identityref {
        base subscription-stream-status;
    }
}
```

```
    }
    description
      "Specifies the status of a subscription or datastream.";
  }

  typedef transport-protocol {
    type identityref {
      base transport;
    }
    description
      "Specifies transport protocol used to send notifications to a
      receiver.";
  }

  typedef push-source {
    type enumeration {
      enum "interface-originated" {
        description
          "Notifications will be sent from a specific interface on
          a publisher";
      }
      enum "address-originated" {
        description
          "Notifications will be sent from a specific address on a
          publisher";
      }
    }
    description
      "Specifies from where notifications will be sourced when
      being sent by the publisher.";
  }

  typedef stream {
    type identityref {
      base stream;
    }
    description
      "Specifies a system-provided datastream.";
  }

  typedef filter-ref {
    type leafref {
      path "/notif-bis:filters/notif-bis:filter/notif-bis:filter-id";
    }
    description
      "This type is used to reference a filter.";
  }
```



```
/*
 * GROUPINGS
 */

grouping base-filter {
  description
    "This grouping defines the base for filters for
    notification events.
    It includes the filter defined in 5277 and
    it enables extending filtering to other
    types of filters";
  choice filter-type {
    description
      "A filter needs to be a single filter of a given type.
      Mixing and matching of multiple filters does not occur
      at the level of this grouping.";
    case rfc5277 {
      anyxml filter {
        description
          "Filter per RFC 5277. Notification filter.
          If a filter element is specified to look for data of a
          particular value, and the data item is not present
          within a particular event notification for its value to
          be checked against, the notification will be filtered
          out. For example, if one were to check for
          'severity=critical' in a configuration event
          notification where this field was not supported, then
          the notification would be filtered out. For subtree
          filtering, a non-empty node set means that the filter
          matches. For XPath filtering, the mechanisms defined
          in [XPATH] should be used to convert the returned
          value to boolean.";
        }
      }
    }
  }
}

grouping subscription-info-basic-non-modifiable {
  description
    "This grouping describes the information in a basic
    subscription request.";
  leaf stream {
    type stream;
    description
      "Indicates which stream of events is of interest.
      If not present, events in the default NETCONF stream
      will be sent.";
  }
}
```

```
        leaf encoding {
          type encoding;
          default "encode-xml";
          description
            "The type of encoding for the subscribed data.
             Default is XML";
        }
      }
    }

    grouping subscription-info-basic-modifiable {
      description
        "This grouping describes some objects which may be changed
         in a subscription via an RPC.";
      uses base-filter;
      leaf startTime {
        type yang:date-and-time;
        description
          "Used to trigger the replay feature
           and indicate that the replay should start at the time
           specified. If <startTime> is not present, this is not a
           replay subscription. It is not valid to specify start
           times that are later than the current time. If the
           <startTime> specified is earlier than the log can support,
           the replay will begin with the earliest available
           notification. This parameter is of type dateTime and
           compliant to [RFC3339]. Implementations must
           support time zones.";
      }
      leaf stopTime {
        type yang:date-and-time;
        must "current() > ../startTime" {
          description
            "stopTime must be used with and be later than <startTime>";
        }
      }
      description
        "Used with the optional replay feature to indicate the
         newest notifications of interest. If <stopTime> is
         not present, the notifications will continue until the
         subscription is terminated. Must be used with and be
         later than <startTime>. Values of <stopTime> in the
         future are valid. This parameter is of type dateTime and
         compliant to [RFC3339]. Implementations must support time
         zones.";
    }
  }
}

grouping subscription-info-all-modifiable {
  description
```

```
    "This grouping describes all rpc modifiable objects in a
    subscription.";
  uses subscription-info-basic-modifiable {
    augment "filter-type" {
      description
        "Post-5277 subscriptions allow references to existing
        filters";
      case by-reference {
        description
          "Incorporate a filter that has been configured
          separately.";
        leaf filter-ref {
          type filter-ref;
          description
            "References filter which is associated with the
            subscription.";
        }
      }
    }
  }
}

grouping subscription-info {
  description
    "This grouping describes information concerning a
    subscription.";
  uses subscription-info-basic-non-modifiable;
  uses subscription-info-all-modifiable;
}

grouping push-source-info {
  description
    "Defines the sender source from which notifications
    for a configured subscription are sent.";
  choice push-source {
    description
      "Identifies the egress interface on the Publisher from
      which notifications will or are being sent.";
    case interface-originated {
      description
        "When the push source is out of an interface on the
        Publisher established via static configuration.";
      leaf source-interface {
        type if:interface-ref;
        description
          "References the interface for notifications.";
      }
    }
  }
}
```

```
case address-originated {
  description
    "When the push source is out of an IP address on the
    Publisher established via static configuration.";
  leaf source-vrf {
    type uint32 {
      range "16..1048574";
    }
    description
      "Label of the vrf.";
  }
  leaf source-address {
    type inet:ip-address-no-zone;
    mandatory true;
    description
      "The source address for the notifications.";
  }
}
}
}

grouping receiver-info {
  description
    "Defines where and how to deliver notifications for a
    configured subscription. This includes
    specifying the receiver, as well as defining
    any network and transport aspects when sending of
    notifications occurs outside of Netconf.";
  container receivers {
    description
      "Set of receivers in a subscription.";
    list receiver {
      key "address";
      min-elements 1;
      description
        "A single host or multipoint address intended as a target
        for the notifications for a subscription.";
      leaf address {
        type inet:host;
        mandatory true;
        description
          "Specifies the address for the traffic to reach a
          remote host. One of the following must be
          specified: an ipv4 address, an ipv6 address,
          or a host name.";
      }
      leaf port {
        type inet:port-number;
      }
    }
  }
}
```

```
        mandatory true;
        description
            "This leaf specifies the port number to use for
            messages destined for a receiver.";
    }
    leaf protocol {
        type transport-protocol;
        default "netconf";
        description
            "This leaf specifies the transport protocol used
            to deliver messages destined for the receiver.";
    }
}
}
}

grouping subscription-response {
    description
        "Defines the output to the rpc's establish-subscription
        and modify-subscription.";
    leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
            "Indicates whether subscription is operational,
            or if a problem was encountered.";
    }
    choice result {
        description
            "Depending on the subscription result, different
            data is returned.";
        case success {
            description
                "This case is used when the subscription request
                was successful and a subscription was created/modified
                as a result";
            leaf subscription-id {
                type subscription-id;
                mandatory true;
                description
                    "Identifier used for this subscription.";
            }
        }
        case no-success {
            description
                "This case applies when a subscription request
                was not successful and no subscription was
                created (or modified) as a result.  In this case,
```

```
        information MAY be returned that indicates
        suggested parameter settings that would have a
        high likelihood of succeeding in a subsequent
        establish-subscription or modify-subscription
        request.";
    uses subscription-info;
}
}
}

/*
 * RPCs
 */

rpc establish-subscription {
    description
        "This RPC allows a subscriber to create
        (and possibly negotiate) a subscription on its own behalf.
        If successful, the subscription
        remains in effect for the duration of the subscriber's
        association with the publisher, or until the subscription
        is terminated by virtue of a delete-subscription request.
        In case an error (as indicated by subscription-result)
        is returned, the subscription is
        not created. In that case, the RPC output
        MAY include suggested parameter settings
        that would have a high likelihood of succeeding in a
        subsequent establish-subscription request.";
    input {
        uses subscription-info;
    }
    output {
        uses subscription-response;
    }
}

rpc create-subscription {
    description
        "This operation initiates an event notification subscription
        that will send asynchronous event notifications to the
        initiator of the command until the association terminates.
        It is not possible to modify or delete a subscription
        that was created using this operation. It is included for
        reasons of backward compatibility with RFC 5277
        implementations.";
    input {
        uses subscription-info-basic-non-modifiable{
            refine "stream" {
```

```
        default "NETCONF";
    }
    }
    uses subscription-info-basic-modifiable;
}

rpc modify-subscription {
  description
    "This RPC allows a subscriber to modify a subscription
    that was previously created using establish-subscription.
    If successful, the subscription
    remains in effect for the duration of the subscriber's
    association with the publisher, or until the subscription
    is terminated by virtue of a delete-subscription request.
    In case an error is returned (as indicated by
    subscription-result), the subscription is
    not modified and the original subscription parameters
    remain in effect. In that case, the rpc error response
    MAY include suggested parameter settings
    that would have a high likelihood of succeeding in a
    subsequent modify-subscription request.";
  input {
    leaf subscription-id {
      type subscription-id;
      description
        "Identifier to use for this subscription.";
    }
    uses subscription-info-all-modifiable;
  }
  output {
    uses subscription-response;
  }
}

rpc delete-subscription {
  description
    "This RPC allows a subscriber to delete a subscription that
    was previously created using establish-subscription.";
  input {
    leaf subscription-id {
      type subscription-id;
      mandatory true;
      description
        "Identifier of the subscription that is to be deleted.
        Only subscriptions that were created using
        establish-subscription can be deleted via this RPC.";
    }
  }
}
```

```
    }
    output {
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription is operational,
           or if a problem was encountered.";
      }
    }
  }
}

/*
 * NOTIFICATIONS
 */

notification replay-complete {
  notif-bis:control-plane-notif;
  description
    "This notification is sent to indicate that all of the
     replay notifications have been sent. It must not be
     sent for any other reason.";
  leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
}

notification notification-complete {
  notif-bis:control-plane-notif;
  description
    "This notification is sent to indicate that a
     subscription, which includes a stop time, has
     finished passing events.";
  leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
}

notification subscription-started {
  notif-bis:control-plane-notif;
  description
```



```
    "This notification indicates that a subscription has
      started and notifications are beginning to be sent.
      This notification shall only be sent to receivers
      of a subscription; it does not constitute a general-purpose
      notification.";
  leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  uses subscription-info;
}

notification subscription-suspended {
  notif-bis:control-plane-notif;
  description
    "This notification indicates that a suspension of the
      subscription by the publisher has occurred. No further
      notifications will be sent until subscription
      resumes.
      This notification shall only be sent to receivers
      of a subscription; it does not constitute a general-purpose
      notification.";
  leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  leaf reason {
    type subscription-susp-reason;
    description
      "Provides a reason for why the subscription was
      suspended.";
  }
}

notification subscription-resumed {
  notif-bis:control-plane-notif;
  description
    "This notification indicates that a subscription that had
      previously been suspended has resumed. Notifications
      will once again be sent.";
  leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
```

```
        "This references the affected subscription.";
    }
}

notification subscription-modified {
    notif-bis:control-plane-notif;
    description
        "This notification indicates that a subscription has
        been modified.  Notifications sent from this point
        on will conform to the modified terms of the
        subscription.";
    leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
    uses subscription-info;
}

notification subscription-terminated {
    notif-bis:control-plane-notif;
    description
        "This notification indicates that a subscription has been
        terminated.";
    leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
    leaf reason {
        type subscription-term-reason;
        description
            "Provides a reason for why the subscription was
            terminated.";
    }
}

/*
 * DATA NODES
 */

container streams {
    config false;
    description
        "This container contains a leaf list of built-in
```

```
        streams that are provided by the system.";
    leaf-list stream {
        type stream;
        description
            "Identifies the built-in streams that are supported by the
            system. Built-in streams are associated with their own
            identities, each of which carries a special semantics.
            In case configurable custom streams are supported,
            as indicated by the custom-stream identity, the
            configuration of those custom streams is provided
            separately.";
    }
}
container filters {
    description
        "This container contains a list of configurable filters
        that can be applied to subscriptions. This facilitates
        the reuse of complex filters once defined.";
    list filter {
        key "filter-id";
        description
            "A list of configurable filters that can be applied to
            subscriptions.";
        leaf filter-id {
            type filter-id;
            description
                "An identifier to differentiate between filters.";
        }
        uses base-filter;
    }
}
container subscription-config {
    if-feature "configured-subscriptions";
    description
        "Contains the list of subscriptions that are configured,
        as opposed to established via RPC or other means.";
    list subscription {
        key "subscription-id";
        description
            "Content of a subscription.";
        leaf subscription-id {
            type subscription-id;
            description
                "Identifier to use for this subscription.";
        }
        uses subscription-info;
        uses receiver-info {
            if-feature "configured-subscriptions";
        }
    }
}
```

```
    }
    uses push-source-info {
      if-feature "configured-subscriptions";
    }
  }
}
container subscriptions {
  config false;
  description
    "Contains the list of currently active subscriptions,
    i.e. subscriptions that are currently in effect,
    used for subscription management and monitoring purposes.
    This includes subscriptions that have been setup via RPC
    primitives, e.g. create-subscription, establish-
    subscription, and modify-subscription, as well as
    subscriptions that have been established via
    configuration.";
  list subscription {
    key "subscription-id";
    config false;
    description
      "Content of a subscription.
      Subscriptions can be created using a control channel
      or RPC, or be established through configuration.";
    leaf subscription-id {
      type subscription-id;
      description
        "Identifier of this subscription.";
    }

    leaf configured-subscription {
      if-feature "configured-subscriptions";
      type empty;
      description
        "The presence of this leaf indicates that the
        subscription originated from configuration, not
        through a control channel or RPC.";
    }

    leaf subscription-status {
      type subscription-status;
      description
        "The status of the subscription.";
    }
    uses subscription-info;
    uses receiver-info {
      if-feature "configured-subscriptions";
    }
  }
}
```

```
    uses push-source-info {  
        if-feature "configured-subscriptions";  
    }  
  }  
}  
<CODE ENDS>
```

9. Backwards Compatibility

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Publishers supporting the features in this document must advertise both capabilities "urn:ietf:params:netconf:capability:notification:1.0" and "urn:ietf:params:netconf:capability:notification:1.1".

An example of a hello message by a publisher during session establishment would be:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <capabilities>  
    <capability>  
      urn:ietf:params:xml:ns:netconf:base:1.0  
    </capability>  
    <capability>  
      urn:ietf:params:netconf:capability:startup:1.0  
    </capability>  
    <capability>  
      urn:ietf:params:netconf:capability:notification:1.0  
    </capability>  
    <capability>  
      urn:ietf:params:netconf:capability:notification:1.1  
    </capability>  
  </capabilities>  
  <session-id>4</session-id>  
</hello>
```

Figure 9: Hello message

Subscribers that only support [RFC5277] recognize capability "urn:ietf:params:netconf:capability:notification:1.0" and ignore capability "urn:ietf:params:netconf:capability:notification:1.1". This allows them interacting with the publisher as per [RFC5277]. Subscribers that support the features in this document recognize both capabilities. This allows them interacting with the publisher as per this document.

Note that to support backwards compatibility, the yang models in this document include two types of naming conventions. That used in [RFC5277], e.g., `replayComplete`; and that commonly used in yang models, e.g., `subscription-started`.

10. Security Considerations

The `<notification>` elements are never sent before the transport layer, including capabilities exchange, has been established and the manager has been securely established.

A secure transport is highly recommended and the publisher must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of content involved. When a `<get>` is received that refers to the content defined in this memo, clients should only be able to view the content for which they have sufficient privileges. `<create-subscription>` and `<establish-subscription>` operations can be considered like deferred `<get>`, and the content that different users can access may vary. This different access is reflected in the `<notification>` to which different users are able to subscribe.

The contents of notifications, as well as the names of event streams, may contain sensitive information and care should be taken to ensure that they are viewed only by authorized users. The publisher **MUST NOT** include any content in a notification that the user is not authorized to view.

If a malicious or buggy subscriber sends a number of `<create-subscription>` requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions can be terminated by terminating the suspect underlying NETCONF sessions using the `<kill-session>` operation. If the subscriber uses `<establish-subscription>`, the publisher can also suspend or terminate subscriptions with per-subscription granularity.

A subscription could be configured on another receiver's behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver, which doesn't even support subscriptions. Subscribers that do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the NETCONF Authorization Control Model [RFC6536] **SHOULD** be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive specific event notification types. The permissions are specified as a set of access control rules.

Note that streams can define additional authorization requirements. For instance, in [I-D.ietf-netconf-yang-push], each of the elements in its data plane notifications must also go through access control.

11. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Yang Geng, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, Michael Scharf, and Guangying Zheng.

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.

12.2. Informative References

[I-D.ietf-netconf-netconf-event-notif]

Gonzalez Prieto, Alberto., Clemm, Alexander., Voit, Eric., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF support for event notifications", August 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>>.

[I-D.ietf-netconf-restconf]

Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-17, September 2016.

[I-D.ietf-netconf-restconf-notif]

Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", August 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf-notif/>>.

[I-D.ietf-netconf-yang-push]

Clemm, Alexander., Gonzalez Prieto, Alberto., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", June 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

Appendix A. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

A.1. Unresolved and yet-to-be addressed issues

EN1 - Definition of basic set of Stream types. What streams are provided and what do they contain (includes default 5277 stream).

EN2 - Clarify interplay between filter definitions and different streams. Includes information in subtrees of event payloads.

EN3 - Mechanisms for diagnostics, e.g. deal with dropped updates, monitoring when they occur, etc

A.2. Agreement in principal

EN4 - How to allow for seamless integration with non-standard encodings and transports (like GPB/GRPC). Specify requirements encoding and transport must meet, provide examples.

EN7 - Detecting loss of a sequential update notification, and mechanisms to resend. Implications to transports must be thought through.

EN6 - Stream discovery. Allow to discover additional stream properties.

EN12 - Test-only option for a subscription is desired. But it still needs to be defined.

EN14 - Ensure that Configured Subscriptions are fully defined in YANG model.

A.3. Resolved Issues

EN5 - This draft obsoletes 5277, as opposed to being in parallel with it

EN8 - No mandatory transport

EN15 - Term for Dynamic and Static Subscriptions (move to "Configured")

EN9 - Multiple receivers per Configured Subscription is ok.

EN13 - RFC6241 Subtree-filter definition in 5277bis cannot apply to elements of an event. Must explicitly define how 6241 doesn't apply filtering within a 5277bis event.

EN10 - Replay support will be provided for selected stream types (modify vs. delete)

EN11 - Required layering security requirements/considerations will be added into the YANG model for Configured Subscriptions. It will be up to the transport to meet these requirements.

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v00 - v01

- o YANG Model changes. New groupings for subscription info to allow restriction of what is changable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.

- o Expanded/renamed definitions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.
- o Removal of redundant with other drafts
- o Many other clean-ups of wording and terminology

Authors' Addresses

Alexander Clemm
Sympotech

Email: alex@sympotech.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Sharon Chisholm
Ciena

Email: schishol@ciena.com

Hector Trevino
Cisco Systems

Email: htrevino@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks
November 3, 2016

SSH Client and Server Models
draft-ietf-netconf-ssh-client-server-01

Abstract

This document defines two YANG modules, one defines groupings for a generic SSH client and the other defines groupings for a generic SSH server. It is intended that these groupings will be used by applications using the SSH protocol.

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. No other RFC Editor instructions are specified elsewhere 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:

- o draft-ietf-netconf-keystore

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

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for draft-ietf-netconf-keystore

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

- o "2016-11-02" --> the publication date of this draft

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

- o Appendix A. Change Log

- o Appendix B. 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 May 7, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Tree Diagrams	3
2. The SSH Client Model	4
2.1. Tree Diagram	4
2.2. Example Usage	6
2.3. YANG Model	6
3. The SSH Server Model	11
3.1. Tree Diagram	11
3.2. Example Usage	12
3.3. YANG Model	13
4. Security Considerations	17

5.	IANA Considerations	17
5.1.	The IETF XML Registry	17
5.2.	The YANG Module Names Registry	17
6.	Acknowledgements	18
7.	References	18
7.1.	Normative References	18
7.2.	Informative References	18
Appendix A.	Change Log	20
A.1.	server-model-09 to 00	20
Appendix B.	Open Issues	20
Authors' Addresses	20

1. Introduction

This document defines two YANG [RFC6020] modules, one defines groupings for a generic SSH client and the other defines groupings for a generic SSH server (SSH is defined in [RFC4252], [RFC4253], and [RFC4254]). It is intended that these groupings will be used by applications using the SSH protocol. For instance, these groupings could be used to help define the data model for an OpenSSH [OPENSSH] server or a NETCONF over SSH [RFC6242] based server.

The two YANG modules in this document each define two groupings. One grouping defines everything other than what's needed for the TCP [RFC793] protocol layer. The other grouping uses the first grouping while adding TCP layer specifics (e.g., addresses to connect to, ports to listen on, etc.). This separation is done in order to enable applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [draft-ietf-netconf-call-home] could use the first grouping for the SSH parts it provides, while adding data nodes for the reversed TCP layer.

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:

- o Brackets "[" and "]" enclose list keys.

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

2. The SSH Client Model

The SSH client model presented in this section contains two YANG groupings, one for a client that initiates the underlying TCP connection and another for a client that has had the TCP connection opened for it already (e.g., call home).

Both of these groupings reference data nodes defined by the Keystore model [draft-ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which trusted CA certificate a client should use to authenticate X.509v3 certificate based host keys [RFC6187].

2.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-ssh-client module.

```

module: ietf-ssh-client
  groupings:
    initiating-ssh-client-grouping
      +----- server-auth
      | +----- trusted-ssh-host-keys?  -> /ks:keystore/trusted-ssh-hos
t-keys/name
      | +----- trusted-ca-certs?      -> /ks:keystore/trusted-certifi
cates/name {ssh-x509-certs}?
      | +----- trusted-server-certs?  -> /ks:keystore/trusted-certifi
cates/name
      +----- client-auth
        +----- matches* [name]
          +----- name?                  string
          +----- match* [name]
            | +----- name?              string
            | +----- trusted-ssh-host-keys?  -> /ks:keystore/trusted-s
sh-host-keys/name
            | +----- trusted-ca-certs?      -> /ks:keystore/trusted-c
ertificates/name
            | +----- trusted-server-certs?  -> /ks:keystore/trusted-c
ertificates/name
            +----- user-auth-credentials?  -> /ks:keystore/user-auth-cr
edentials/user-auth-credential/username

    listening-ssh-client-grouping
      +----- address?      inet:ip-address
      +----- port?        inet:port-number
      +----- server-auth
      | +----- trusted-ssh-host-keys?  -> /ks:keystore/trusted-ssh-hos
t-keys/name
      | +----- trusted-ca-certs?      -> /ks:keystore/trusted-certifi
cates/name {ssh-x509-certs}?
      | +----- trusted-server-certs?  -> /ks:keystore/trusted-certifi
cates/name
      +----- client-auth
        +----- matches* [name]
          +----- name?                  string
          +----- match* [name]
            | +----- name?              string
            | +----- trusted-ssh-host-keys?  -> /ks:keystore/trusted-s
sh-host-keys/name
            | +----- trusted-ca-certs?      -> /ks:keystore/trusted-c
ertificates/name
            | +----- trusted-server-certs?  -> /ks:keystore/trusted-c
ertificates/name
            +----- user-auth-credentials?  -> /ks:keystore/user-auth-cr
edentials/user-auth-credential/username

```


2.2. Example Usage

This section shows how it would appear if the `initiating-ssh-client-grouping` were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

FIXME (how to do an example for a module that only has groupings?)

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [draft-ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-ssh-client@2016-11-02.yang"

module ietf-ssh-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-client";
  prefix "sshc";

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

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
  }

  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>
```

Author: Kent Watsen
<mailto:kwatsen@juniper.net>

Author: Gary Wu
<mailto:garywu@cisco.com>;

description

"This module defines a reusable grouping for a SSH client that can be used as a basis for specific SSH client 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 XXXX; see the RFC itself for full legal notices."

```
revision "2016-11-02" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: SSH Client and Server Models";  
}
```

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

```
grouping initiating-ssh-client-grouping {  
  description  
    "A reusable grouping for a SSH client that initiates the  
    underlying TCP transport connection."  
  
  container server-auth {  
    description  
      "Trusted server identities.";
```

```
leaf trusted-ssh-host-keys {
  type leafref {
    path "/ks:keystore/ks:trusted-ssh-host-keys/ks:name";
  }
  description
    "A reference to a list of SSH host keys used by the
    SSH client to authenticate SSH server host keys.
    A server host key is authenticate if it is an exact
    match to a configured trusted SSH host key.";
}

leaf trusted-ca-certs {
  if-feature ssh-x509-certs;
  type leafref {
    path "/ks:keystore/ks:trusted-certificates/ks:name";
  }
  description
    "A reference to a list of certificate authority (CA)
    certificates used by the SSH client to authenticate
    SSH server certificates.";
}

leaf trusted-server-certs {
  type leafref {
    path "/ks:keystore/ks:trusted-certificates/ks:name";
  }
  description
    "A reference to a list of server certificates used by
    the SSH client to authenticate SSH server certificates.
    A server certificate is authenticated if it is an
    exact match to a configured trusted server certificate.";
}
}

container client-auth {
  description
    "The credentials used by the client to authenticate to
    the SSH server.";

  list matches {
    key name;
    description
      "A matches expression, which performs like a firewall
      rulebase in that each matches expression is considered
      for a match before moving onto the next matches
      expression. The first matching expression terminates
      the search, and its 'user-auth-credentials' are used
      to log into the SSH server.";
```

```
leaf name {
  type string;
  description
    "An arbitrary name for this matches expression.";
}
list match {
  key name;
  description
    "A match rule. The presented SSH server's host key
    is matched against possible trusted SSH host keys
    and certificates. If a match is found, the specified
    'user-auth-credentials' is used to log into the
    SSH server.";
  leaf name {
    type string;
    description
      "An arbitrary name for this match rule.";
  }
  leaf trusted-ssh-host-keys {
    type leafref {
      path "/ks:keystore/ks:trusted-ssh-host-keys/ks:name";
    }
    description
      "A test to see if the presented SSH host key
      matches any of the host keys in the specified
      'trusted-ssh-host-keys' list maintained by the
      keystore module.";
  }
  leaf trusted-ca-certs {
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A test to see if the presented SSH host key matches
      any of the trusted CA certificates in the specified
      'trusted-certificates' list maintained by the
      keystore module.";
  }
  leaf trusted-server-certs {
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A test to see if the presented SSH host key matches
      any of the trusted server certificates in the specified
      'trusted-certificates' list maintained by the
      keystore module.";
  }
}
```

```
    }
    leaf user-auth-credentials {
      type leafref {
        path "/ks:keystore/ks:user-auth-credentials/"
          + "ks:user-auth-credential/ks:username";
      }
      description
        "The specific user authentication credentials to use if
        all of the above 'match' expressions match.";
    }
  }
} // end initiating-ssh-client-grouping

grouping listening-ssh-client-grouping {
  description
    "A reusable grouping for a SSH client that does not
    the underlying TCP transport connection have been
    established using some other mechanism.";
  leaf address {
    type inet:ip-address;
    description
      "The IP address to listen for call-home connections on.";
  }
  leaf port {
    type inet:port-number;
    description
      "The port number to listen for call-home connections.
      When this grouping is used, it is RECOMMENDED that
      refine statement is used to either set a default port
      value or to set mandatory true.";
  }
  uses initiating-ssh-client-grouping;
}

}
```

<CODE ENDS>

3. The SSH Server Model

The SSH server model presented in this section contains two YANG groupings, one for a server that opens a socket to accept TCP connections and another for a server that has had the TCP connection opened for it already (e.g., `inetd`).

Both of these groupings reference data nodes defined by the Keystore model [draft-ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which host key a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the `ietf-ssh-server` module.

```

module: ietf-ssh-server
groupings:
  listening-ssh-server-grouping
    +----- address?          inet:ip-address
    +----- port?             inet:port-number
    +----- host-keys
      |
      | +----- host-key* [name]
      | | +----- name?          string
      | | +----- (host-key-type)
      | | | +---:(public-key)
      | | | | +----- public-key?    -> /ks:keystore/private-keys/pri
vate-key/name
      | | | | +---:(certificate)
      | | | | | +----- certificate?  -> /ks:keystore/private-keys/pri
vate-key/certificate-chains/certificate-chain/name {ssh-x509-certs}?
      | | +----- client-cert-auth {ssh-x509-certs}?
      | | +----- trusted-ca-certs?    -> /ks:keystore/trusted-certific
ates/name
      | | +----- trusted-client-certs? -> /ks:keystore/trusted-certific
ates/name
      |
      | non-listening-ssh-server-grouping
      | +----- host-keys
      | | +----- host-key* [name]
      | | | +----- name?          string
      | | | +----- (host-key-type)
      | | | | +---:(public-key)
      | | | | | +----- public-key?    -> /ks:keystore/private-keys/pri
vate-key/name
      | | | | +---:(certificate)
      | | | | | +----- certificate?  -> /ks:keystore/private-keys/pri
vate-key/certificate-chains/certificate-chain/name {ssh-x509-certs}?
      | | +----- client-cert-auth {ssh-x509-certs}?
      | | +----- trusted-ca-certs?    -> /ks:keystore/trusted-certific
ates/name
      | | +----- trusted-client-certs? -> /ks:keystore/trusted-certific
ates/name

```

3.2. Example Usage

This section shows how it would appear if the listening-ssh-server-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

```
<listening-ssh-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">
  <port>830</port>
  <host-keys>
    <host-key>
      <name>deployment-specific-certificate</name>
      <certificate>ex-key-sect571r1-cert</certificate>
    </host-key>
  </host-keys>
</certificates>
<client-cert-auth>
  <trusted-ca-certs>
    deployment-specific-ca-certs
  </trusted-ca-certs>
  <trusted-client-certs>
    explicitly-trusted-client-certs
  </trusted-client-certs>
</client-cert-auth>
</listening-ssh-server>
```

3.3. YANG Model

This YANG module has a normative references to [RFC4253], [RFC6991], and [draft-ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-ssh-server@2016-11-02.yang"

module ietf-ssh-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
  prefix "sshs";

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

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
  }

  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 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 XXXX; see the RFC itself for full legal notices."

```
revision "2016-11-02" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: SSH Client and Server 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 non-listening-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 host keys the SSH server will use to
        construct its ordered list of algorithms, when sending
        its SSH_MSG_KEXINIT message, as defined in Section 7.1
        of RFC 4253.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
      leaf name {
        type string;
        description
          "An arbitrary name for this host-key";
      }
      choice host-key-type {
        mandatory true;
        description
          "The type of host key being specified";
        leaf public-key {
          type leafref {
            path "/ks:keystore/ks:private-keys/ks:private-key/"
              + "ks:name";
          }
          description
            "The public key is actually identified by the name of
            its cooresponding private-key in the keystore.";
        }
        leaf certificate {
          if-feature ssh-x509-certs;
          type leafref {
            path "/ks:keystore/ks:private-keys/ks:private-key/"
              + "ks:certificate-chains/ks:certificate-chain/"
              + "ks:name";
          }
          description
            "The name of a certificate in the keystore.";
        }
      }
    }
  }
}
```

```
    }
  }
}

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 "/ks:keystore/ks:trusted-certificates/ks: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 "/ks:keystore/ks:trusted-certificates/ks: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.";
  }
}

grouping listening-ssh-server-grouping {
  description
    "A reusable grouping for a SSH server that can be used as a
    basis for specific SSH server instances.";
  leaf address {
    type inet:ip-address;
    description
      "The IP address of the interface to listen on. The SSH
      server will listen on all interfaces if no value is
      specified. Please note that some addresses have special
      meanings (e.g., '0.0.0.0' and '::').";
  }
  leaf port {
```

```
    type inet:port-number;
    description
      "The local port number on this interface the SSH server
       listens on. When this grouping is used, it is RECOMMENDED
       that refine statement is used to either set a default port
       value or to set mandatory true.";
  }
  uses non-listening-ssh-server-grouping;
}
```

<CODE ENDS>

4. Security Considerations

5. IANA Considerations

5.1. The IETF XML Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

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

name:	ietf-ssh-client
namespace:	urn:ietf:params:xml:ns:yang:ietf-ssh-client
prefix:	sshc
reference:	RFC XXXX
name:	ietf-ssh-server
namespace:	urn:ietf:params:xml:ns:yang:ietf-ssh-server
prefix:	sshs
reference:	RFC XXXX

6. 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, Juergen Schoenwaelder, Phil Shafer, Sean Turner, Michal Vasko, and Bert Wijnen.

7. References

7.1. Normative References

- [draft-ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), 2016,
<<https://datatracker.ietf.org/html/draft-ietf-netconf-keystore>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010,
<<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, DOI 10.17487/RFC6187, March 2011, <<http://www.rfc-editor.org/info/rfc6187>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013,
<<http://www.rfc-editor.org/info/rfc6991>>.

7.2. Informative References

- [draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), 2015,
<<https://datatracker.ietf.org/html/draft-ietf-netconf-call-home-17>>.
- [OPENSSH] "OpenSSH", 2016, <<http://www.openssh.com>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <<http://www.rfc-editor.org/info/rfc4252>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.
- [RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <<http://www.rfc-editor.org/info/rfc4254>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC793] Postel, J., "TRANSMISSION CONTROL PROTOCOL", STD 7, September 1981, <<https://www.ietf.org/rfc/rfc793.txt>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Added in previously missing ietf-ssh-client module.
- o Noted that '0.0.0.0' and ':::' might have special meanings.

Appendix B. Open Issues

Please see: <https://github.com/netconf-wg/ssh-client-server/issues>.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

K. Watsen
Juniper Networks
November 3, 2016

TLS Client and Server Models
draft-ietf-netconf-tls-client-server-01

Abstract

This document defines two YANG modules, one defines groupings for a generic TLS client and the other defines groupings for a generic TLS server. It is intended that these groupings will be used by applications using the TLS protocol.

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. No other RFC Editor instructions are specified elsewhere 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:

- o draft-ietf-netconf-keystore

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

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for draft-ietf-netconf-keystore

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

- o "2016-11-02" --> the publication date of this draft

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

- o Appendix A. Change Log
- o Appendix B. 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 May 7, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Tree Diagrams	3
2. The TLS Client Model	4
2.1. Tree Diagram	4
2.2. Example Usage	4
2.3. YANG Model	5
3. The TLS Server Model	7
3.1. Tree Diagram	7
3.2. Example Usage	8
3.3. YANG Model	8
4. Security Considerations	11
5. IANA Considerations	11
5.1. The IETF XML Registry	11

5.2. The YANG Module Names Registry	12
6. Acknowledgements	12
7. References	12
7.1. Normative References	12
7.2. Informative References	13
Appendix A. Change Log	14
A.1. server-model-09 to 00	14
Appendix B. Open Issues	14
Author's Address	14

1. Introduction

This document defines two YANG [RFC6020] modules, one defines groupings for a generic TLS client and the other defines groupings for a generic TLS server (TLS is defined in [RFC5246]). It is intended that these groupings will be used by applications using the TLS protocol. For instance, these groupings could be used to help define the data model for an HTTPS [RFC2818] server or a NETCONF over TLS [RFC7589] based server.

The two YANG modules in this document each define two groupings. One grouping defines everything other than what's needed for the TCP [RFC793] protocol layer. The other grouping uses the first grouping while adding TCP layer specifics (e.g., addresses to connect to, ports to listen on, etc.). This separation is done in order to enable applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [draft-ietf-netconf-call-home] could use the first grouping for the TLS parts it provides, while adding data nodes for the reversed TCP layer.

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:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.

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

2. The TLS Client Model

EDITOR NOTE: Please ignore this section, it is incomplete.

The TLS client model presented in this section contains two YANG groupings, one for a client that initiates the underlying TCP connection and another for a client that has had the TCP connection opened for it already (e.g., call home).

Both of these groupings reference data nodes defined by the Keystore model [draft-ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which trusted CA certificate a client should use to authenticate the server's certificate.

2.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-tls-client module.

```
module: ietf-tls-client
  groupings:
    initiating-tls-client-grouping
      +---- some-TBD-tcp-client-stuff?   string
      +---- some-TBD-tls-client-stuff?   string

    non-initiating-tls-client-grouping
      +---- some-TBD-tls-client-stuff?   string
```

2.2. Example Usage

This section shows how it would appear if the initiating-tls-client-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

FIXME

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [draft-ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-tls-client@2016-11-02.yang"

// Editor's Note:
// This module is incomplete at this time. Below is
// just a skeleton so there's something in the draft.
// Please ignore this module for now!

module ietf-tls-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-client";
  prefix "tlsc";
/*
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
  }
*/
  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 defines a reusable grouping for a TLS client that can be used as a basis for specific TLS client 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 XXXX; see the RFC itself for full legal notices."

```
revision "2016-11-02" {
  description
    "Initial version";
  reference
    "RFC XXXX: TLS Client and Server Models";
}

grouping initiating-tls-client-grouping {
  description
    "A reusable grouping for a TLS client that initiates the
    underlying TCP transport connection.";
  leaf some-TBD-tcp-client-stuff {
    type string;
    description "";
  }
  uses non-initiating-tls-client-grouping;
}

grouping non-initiating-tls-client-grouping {
  description
    "A reusable grouping for a TLS client that does not initiate
    the underlying TCP transport connection.";
  leaf some-TBD-tls-client-stuff {
    type string;
    description "";
  }
}
}
```

<CODE ENDS>

3. The TLS Server Model

The TLS server model presented in this section contains two YANG groupings, one for a server that opens a socket to accept TCP connections and another for a server that has had the TCP connection opened for it already (e.g., inetd).

Both of these groupings reference data nodes defined by the Keystore model [draft-ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate the certificate a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-tls-server module.

```

module: ietf-tls-server
  groupings:
    listening-tls-server-grouping
      +---- address?          inet:ip-address
      +---- port?            inet:port-number
      +---- certificates
      |   +---- certificate* [name]
      |   |   +---- name?    -> /ks:keystore/private-keys/private-key/cert
      |   |   ificate-chains/certificate-chain/name
      |   +---- client-auth
      |   |   +---- trusted-ca-certs?      -> /ks:keystore/trusted-certific
      |   |   ificates/name
      |   |   +---- trusted-client-certs?  -> /ks:keystore/trusted-certific
      |   |   ificates/name
      non-listening-tls-server-grouping
        +---- certificates
        |   +---- certificate* [name]
        |   |   +---- name?    -> /ks:keystore/private-keys/private-key/cert
        |   |   ificate-chains/certificate-chain/name
        |   +---- client-auth
        |   |   +---- trusted-ca-certs?      -> /ks:keystore/trusted-certific
        |   |   ificates/name
        |   |   +---- trusted-client-certs?  -> /ks:keystore/trusted-certific
        |   |   ificates/name

```

3.2. Example Usage

This section shows how it would appear if the listening-tls-server-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-keystore].

```
<listening-tls-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <port>6513</port>
  <certificates>
    <certificate>
      <name>ex-key-sect571r1-cert</name>
    </certificate>
  </certificates>
  <client-auth>
    <trusted-ca-certs>
      deployment-specific-ca-certs
    </trusted-ca-certs>
    <trusted-client-certs>
      explicitly-trusted-client-certs
    </trusted-client-certs>
  </client-auth>
</listening-tls-server>
```

3.3. YANG Model

This YANG module has a normative references to [RFC6991], and [draft-ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-tls-server@2016-11-02.yang"

module ietf-tls-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
  prefix "tlss";

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

  import ietf-keystore {
    prefix ks;
    reference
```

```
    "RFC YYYY: Keystore Model";
}

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 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 XXXX; see
  the RFC itself for full legal notices.";

revision "2016-11-02" {
  description
    "Initial version";
  reference
    "RFC XXXX: TLS Client and Server Models";
}

// grouping
grouping non-listening-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 in its Certificate message,
    as defined in Section 7.4.2 in RFC 5246.";
  reference
    "RFC 5246:
    The Transport Layer Security (TLS) Protocol Version 1.2";
  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 "/ks:keystore/ks:private-keys/ks:private-key/"
          + "ks:certificate-chains/ks:certificate-chain/"
          + "ks:name";
      }
      description
        "The name of the certificate in the keystore.";
    }
  }
}

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 "/ks:keystore/ks:trusted-certificates/ks: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 "/ks:keystore/ks:trusted-certificates/ks:name";
    }
  }
}
```

```
    }
    description
        "A reference to a list of client certificates used by
        the TLS server to authenticate TLS client certificates.
        A clients certificate is authenticated if it is an
        exact match to a configured trusted client certificate.";
    }
}
}
```

```
grouping listening-tls-server-grouping {
    description
        "A reusable grouping for a TLS server that can be used as a
        basis for specific TLS server instances.";
    leaf address {
        type inet:ip-address;
        description
            "The IP address of the interface to listen on. The TLS
            server will listen on all interfaces if no value is
            specified. Please note that some addresses have special
            meanings (e.g., '0.0.0.0' and '::').";
    }
    leaf port {
        type inet:port-number;
        description
            "The local port number on this interface the TLS server
            listens on. When this grouping is used, it is RECOMMENDED
            that refine statement is used to either set a default port
            value or to set mandatory true.";
    }
    uses non-listening-tls-server-grouping;
}
```

<CODE ENDS>

4. Security Considerations

5. IANA Considerations

5.1. The IETF XML Registry

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

URI: urn:ietf:params:xml:ns:yang:ietf-tls-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tls-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

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

name:	ietf-tls-client
namespace:	urn:ietf:params:xml:ns:yang:ietf-tls-client
prefix:	tlsc
reference:	RFC XXXX
name:	ietf-tls-server
namespace:	urn:ietf:params:xml:ns:yang:ietf-tls-server
prefix:	tlss
reference:	RFC XXXX

6. 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, Juergen Schoenwaelder, Phil Shafer, Sean Turner, and Bert Wijnen.

7. References

7.1. Normative References

- [draft-ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), 2016,
<<https://datatracker.ietf.org/html/draft-ietf-netconf-keystore>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.

7.2. Informative References

- [draft-ietf-netconf-call-home] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), 2015, <<https://datatracker.ietf.org/html/draft-ietf-netconf-call-home-17>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC793] Postel, J., "TRANSMISSION CONTROL PROTOCOL", STD 7, September 1981, <<https://www.ietf.org/rfc/rfc793.txt>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Noted that '0.0.0.0' and ':::' might have special meanings.

Appendix B. Open Issues

Please see: <https://github.com/netconf-wg/tls-client-server/issues>.

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 1, 2017

Pre-release version

A. Clemm
Sympotech
E. Voit
A. Gonzalez Prieto
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Bierman
YumaWorks
B. Lengyel
Ericsson
October 28, 2016

Subscribing to YANG datastore push updates
draft-ietf-netconf-yang-push-04

Abstract

This document defines a subscription and push mechanism for YANG datastores. This mechanism allows subscriber applications to request updates from a YANG datastore, which are then pushed by the publisher to a receiver per a subscription policy, without requiring additional subscriber requests.

Status of This Memo

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

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

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

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

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
2. Definitions and Acronyms	5
3. Solution Overview	6
3.1. Subscription Model	6
3.2. Negotiation of Subscription Policies	8
3.3. On-Change Considerations	9
3.4. Data Encodings	10
3.5. YANG object filters	11
3.6. Push Data Stream and Transport Mapping	11
3.7. Subscription management	15
3.8. Other considerations	16
4. A YANG data model for management of datastore push subscriptions	20
4.1. Overview	20
4.2. Update streams	26
4.3. Filters	27
4.4. Subscription configuration	27
4.5. Subscription monitoring	29
4.6. Notifications	29
4.7. RPCs	31
5. YANG module	35
6. Security Considerations	47
7. Acknowledgments	48
8. References	48

8.1. Normative References	48
8.2. Informative References	48
Appendix A. Issues that are currently being worked and resolved	49
A.1. Unresolved and yet-to-be addressed issues	49
A.2. Agreement in principal	49
Appendix B. Changes between revisions	50
Authors' Addresses	50

1. Introduction

YANG [RFC7950] was originally designed for the Netconf protocol [RFC6241] which focused on configuration data. However, YANG can be used to model both configuration and operational data. It is therefore reasonable to expect YANG datastores will increasingly be used to support applications that care about both.

For example, service assurance applications will need to be aware of any remote updates to configuration and operational objects. Rapid awareness of object changes will enable such things as validating and maintaining cross-network integrity and consistency, or monitoring state and key performance indicators of remote devices.

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

- o It introduces additional load on network, devices, and applications. Each polling cycle requires a separate yet arguably redundant request that results in an interrupt, requires parsing, consumes bandwidth.
- o It lacks robustness. Polling cycles may be missed, requests may be delayed or get lost, often particularly in cases when the network is under stress and hence exactly when the need for the data is the greatest.
- o Data may be difficult to calibrate and compare. Polling requests may undergo slight fluctuations, resulting in intervals of different lengths which makes data hard to compare. Likewise, pollers may have difficulty issuing requests that reach all devices at the same time, resulting in offset polling intervals which again make data hard to compare.

A more effective alternative to polling is when an application can request to be automatically updated on current relevant content of a datastore. If such a request is accepted, interesting updates will subsequently be pushed from that datastore.

Dependence on polling-based management is typically considered an important shortcoming of applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores will be any more effective, as they would follow the same request/response pattern.

While YANG allows the definition of push notifications, such notifications generally indicate the occurrence of certain well-specified event conditions, such as the onset of an alarm condition or the occurrence of an error. A capability to subscribe to and deliver such pre-defined event notifications has been defined in [RFC5277]. In addition, configuration change notifications have been defined in [RFC6470]. These change notifications pertain only to configuration information, not to operational state, and convey the root of the subtree to which changes were applied along with the edits, but not the modified data nodes and their values. Furthermore, while delivery of updates using notifications is a viable option, some applications desire the ability to stream updates using other transports.

Accordingly, there is a need for a service that allows applications to dynamically subscribe to updates of a YANG datastore and that allows the publisher to push those updates, possibly using one of several delivery mechanisms. Additionally, support for subscriptions configured directly on the publisher are also useful when dynamic signaling is undesirable or unsupported. The requirements for such a service are documented in [RFC7923].

This document proposes a solution. The solution builds on top of the Netconf Event Model [I-D.ietf-netconf-5277bis] which defines a mechanism for the management of event subscriptions. At its core, the solution defined here introduces a new set of event streams which maybe subscribed, introduces datastore push update mechanisms, and provides extensions to the event subscription model. The document also includes YANG data model augmentations which extend the model and RPCs defined within [I-D.ietf-netconf-5277bis].

Key capabilities worth highlighting include:

- o An extension to event subscription mechanisms allowing clients to subscribe to event streams containing automatic datastore updates. The subscription allows clients to specify which data they are interested in, what types of updates (e.g. create, delete, modify), and to provide optional filters with criteria that data must meet for updates to be sent. Furthermore, subscriptions can specify a policy that directs when updates are provided. For

example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

- o Format and contents of the YANG push updates themselves.
- o The ability for a publisher to push back on requested subscription parameters. Because not every publisher may support every requested update policy for every piece of data, it is necessary for a publisher to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate push update subscription parameters. For example, some publishers may have a lower limit to the period with which they can send updates, or they may not support on-change updates for every piece of data.
- o Subscription parameters which allow the specification of QoS extensions to address prioritization between independent streams of updates.

2. Definitions and Acronyms

Many of the terms in this document are defined in [I-D.ietf-netconf-5277bis]. Please see that document for these definitions.

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

Data node update: A data item containing the current value/property of a Data node at the time the data node update was created.

Data record: A record containing a set of one or more data node instances and their associated values.

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

Datastream: A continuous stream of data records, each including a set of updates, i.e. data node instances and their associated values.

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

Push-update stream: A conceptual data stream of a datastore that streams the entire datastore contents continuously and perpetually.

Update: A data item containing the current value of a data node.

Update notification: An Event Notification including those data node update(s) to be pushed in order to meet the obligations of a single Subscription. All included data node updates must reflect the state of a Datastore at a snapshot in time.

Update record: A representation of a data node update as a data record. An update record can be included as part of an update stream. It can also be logged for retrieval. In general, an update record will include the value/property of a data node. It may also include information about the type of data node update, i.e. whether the data node was modified/updated, or newly created, or deleted.

Update trigger: A mechanism, as specified by a Subscription Policy, that determines when a data node update is to be communicated. (e.g., a change trigger, invoked when the value of a data node changes or a data node is created or deleted, or a time trigger, invoked after the laps of a periodic time interval.)

YANG object filter: A filter that contains evaluation criteria which are evaluated against YANG objects of a subscription. An update is only published if the object meets the specified filter criteria.

YANG-Push: The subscription and push mechanism for YANG datastores that is specified in this document.

3. Solution Overview

This document specifies a solution for a push update subscription service. This solution supports the dynamic as well as configured subscriptions to information updates from YANG datastores. A subscription might target exposed operational and/or configuration YANG objects on a device. YANG objects are subsequently pushed from the publisher to the receiver per the terms of the subscription.

3.1. Subscription Model

YANG-push subscriptions are defined using a data model that is itself defined in YANG. This model augments the event subscription model defined in [I-D.ietf-netconf-5277bis] and introduces several new parameters that allow subscribers to specify what to include in an update notification and what triggers such an update notification.

The subscription model assumes the presence of one or more conceptual perpetual datastreams of continuous subscribable YANG updates. There are several datastreams with predefined semantics, such as the stream of updates of all operational data or the stream of updates of all config data. In addition, it is possible to define custom streams with customizable semantics. The model includes the list of update

datastreams that are supported by a system and available for subscription.

The subscription model augments the [I-D.ietf-netconf-5277bis] subscription model with a set of parameters:

- o Anydata encoding for periodic and on-change push updates.
- o A subscription policy definition regarding the update trigger when to send new update notifications.
- * For periodic subscriptions, the trigger is defined by two parameters that defines the interval with which updates are to be pushed. These parameters are the period/interval of reporting duration, and an anchor time which can be used to calculate at which times updates needs to be assembled and sent.
- * EDITOR'S NOTE: A possible option to discuss concerns the introduction of an additional parameter "changes-only" for periodic subscription. Including this flag would results in sending at the end of each period an update containing only changes since the last update (i.e. a change-update as in the case of an on-change subscription), not a full snapshot of the subscribed information. Such an option might be interesting in case of data that is largely static and bandwidth-constrained environments.
- * For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that can be guided by additional parameters. Please refer also to Section 3.3.
- + One parameter specifies the dampening period, i.e. the interval that must pass before a successive update notification for the same Subscription is sent. Note that the dampening period applies to the set of all data nodes within a single subscription. This means that on the first change of an object, an update notification containing that object is sent either immediately or at the end of a dampening period already in effect.
- + Another parameter allows to restrict the types of changes for which updates are sent (changes to object values, object creation or deletion events). It is conceivable to augment the data model with additional parameters in the future to specify even more refined policies, such as parameters that

specify the magnitude of a change that must occur before an update is triggered.

- + A third parameter specifies whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription to facilitate synchronization and establish the frame of reference for subsequent updates.
- o Optionally, a filter, or set of filters, describing the subset of data node updates that are of interest to the subscriber. The publisher must only send to the subscriber those data node updates that can traverse applied filter(s). The absence of a filter indicates that all data items from the stream are of interest to the subscriber and all data records must be sent in their entirety to the subscriber. The following types of filters are supported: subtree filters, with the same semantics as defined in [RFC6241][RFC6241], and XPath filters. Additional filter types can be added through augmentations. Filters can be specified "inline" as part of the subscription, or can be configured separately and referenced by a subscription, in order to facilitate reuse of complex filters.

The subscription data model is specified as part of the YANG data model described later in this specification. It is conceivable that additional subscription parameters might be added in the future. This can be accomplished through augmentation of the subscription data model.

3.2. Negotiation of Subscription Policies

Dynamic subscriptions must support a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is limited to a single pair of subscription request and response messages. For negative response messages, the publisher SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters constitute suggestions that, when followed, increase the likelihood of success for subsequent requests. However, there are no guarantee that subsequent requests for this subscriber will in fact be accepted.

A subscription request might be declined based on publisher's assessment that it may be unable to provide a filtered update notification stream that would meet the terms of the establish-subscription request.

In case a subscriber requests an encoding other than XML, and this encoding is not supported by the publisher, the publisher simply indicates in the response that the encoding is not supported.

A subscription negotiation capability has been introduced as part of the NETCONF Event Notifications model. However, the ability to negotiate subscriptions is of particular importance in conjunction with push updates, as publisher implementations may have limitations with regards to what updates can be generated and at what velocity.

3.3. On-Change Considerations

On-change subscriptions allow subscribers to subscribe to updates whenever changes to objects occur. As such, on-change subscriptions are of particular interest for data that changes relatively infrequently, yet that require applications to be notified with minimal delay when changes do occur.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Specifically, on-change subscriptions may involve a notion of state to see if a change occurred between past and current state, or the ability to tap into changes as they occur in the underlying system. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

When an on-change subscription is requested for a datastream with a given subtree filter, where not all objects support on-change update triggers, the subscription request **MUST** be rejected. As a result, on-change subscription requests will tend to be directed at very specific, targeted subtrees with only few objects.

Any updates for an on-change subscription will include only objects for which a change was detected. To avoid flooding receivers with repeated updates for fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update for a given object is sent, no other updates for this particular object are sent until the end of the dampening period. Values sent at the end of the dampening period are the values current when that dampening period expires. In addition, updates include information about objects that were deleted and ones that were newly created.

On-change subscriptions can be refined to let users subscribe only to certain types of changes, for example, only to object creations and deletions, but not to modifications of object values.

Additional refinements are conceivable. For example, in order to avoid sending updates on objects whose values undergo only a

negligible change, additional parameters might be added to an on-change subscription specifying a YANG object filter that states how large or "significant" a change has to be before an update is sent. A simple policy is a "delta-policy" that states, for integer-valued data nodes, the minimum difference between the current value and the value that was last reported that triggers an update. Also more sophisticated policies are conceivable, such as policies specified in percentage terms or policies that take into account the rate of change. While not specified as part of this draft, such policies can be accommodated by augmenting the subscription data model accordingly.

3.4. Data Encodings

Subscribed data is encoded in either XML or JSON format. A publisher MUST support XML encoding and MAY support JSON encoding.

It is conceivable that additional encodings may be supported as options in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.

3.4.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been simply retrieved using a get operation and is encoded in the same way. XML encoding rules for data nodes are defined in [RFC7950]. JSON encoding rules are defined in [RFC7951]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

3.4.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed data nodes but also the types of changes that occurred since the last update, such as whether data nodes were newly created since the last update or whether they were merely modified, as well as which data nodes were deleted.

Encoding rules for data in on-change updates correspond to how data would be encoded in a YANG-patch operation as specified in [I-D.ietf-netconf-yang-patch]. The "YANG-patch" would in this case be applied to the earlier state reported by the preceding update, to result in the now-current state of YANG data. Of course, contrary to a YANG-patch operation, the data is sent from the publisher to the receiver and is not restricted to configuration data.

3.5. YANG object filters

Subscriptions can specify filters for subscribed data. The following filters are supported:

- o subtree-filter: A subtree filter specifies a subtree that the subscription refers to. When specified, updates will only concern data nodes from this subtree. Syntax and semantics correspond to that specified for [RFC6241] section 6.
- o xpath-filter: An XPath filter specifies an XPath expression applied to the data in an update, assuming XML-encoded data.

Only a single filter can be applied to a subscription at a time.

It is conceivable for implementations to support other filters. For example, an on-change filter might specify that changes in values should be sent only when the magnitude of the change since previous updates exceeds a certain threshold. It is possible to augment the subscription data model with additional filter types.

3.6. Push Data Stream and Transport Mapping

Pushing data based on a subscription could be considered analogous to a response to a data retrieval request, e.g. a "get" request. However, contrary to such a request, multiple responses to the same request may get sent over a longer period of time.

An applicable mechanism is that of a notification. There are however some specifics that need to be considered. Contrary to other notifications that are associated with alarms and unexpected event occurrences, update notifications are solicited, i.e. tied to a particular subscription which triggered the notification.

A push update notification contains several parameters:

- o A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.
- o Data nodes containing a representation of the datastore subtree(s) containing the updates. In all cases, the subtree(s) are filtered per access control rules to contain only data that the subscriber is authorized to see. For on-change subscriptions, the subtree may only contain the data nodes which have changed since the start of the previous dampening interval.

This document introduces two generic notifications: "push-update" and "push-change-update". Those notifications may be encapsulated on a

transport (e.g. Netconf notifications and HTTP) to carry data records with updates of datastore contents as specified by a subscription. It is possible also map notifications to other transports and encodings and use the same subscription model; however, the definition of such mappings is outside the scope of this document.

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

The contents of the notification conceptually represents the union of all data nodes in the yang modules supported by the publisher. However, in a YANG data model, it is not practical to model the precise data contained in the updates as part of the notification. This is because the specific data nodes supported depend on the implementing system and may even vary dynamically. Therefore, to capture this data, a single parameter that can represent any datastore contents is used, not parameters that represent data nodes one at a time.

A push-change-update notification is the most common type of update for on-change subscriptions. It is not used for periodic subscriptions. The update record in this case contains a data snippet that indicates the full set of changes that data nodes have undergone since the last notification of YANG objects. In other words, this indicates which data nodes have been created, deleted, or have had changes to their values. The format of the data snippet follows YANG-patch [I-D.ietf-netconf-yang-patch], i.e. the same format that would be used with a YANG-patch operation to apply changes to a data tree, indicating the creates, deletes, and modifications of data nodes. Please note that as the update can include a mix of configuration and operational data

The following is an example of push notification. It contains an update for subscription 1011, including a subtree with root foo that contains a leaf, bar:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-update
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>1011</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-contents-xml>
      <foo>
        <bar>some_string</bar>
      </foo>
    </datastore-contents-xml>
  </push-update>
</notification>
```

Figure 1: Push example

The following is an example of an on-change notification. It contains an update for subscription 89, including a new value for a leaf called beta, which is a child of a top-level container called alpha:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-change-update xmlns=
    "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>89</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-changes-xml>
      <alpha xmlns="http://example.com/sample-data/1.0" >
        <beta>1500</beta>
      </alpha>
    </datastore-changes-xml>
  </push-change-update>
</notification>
```

Figure 2: Push example for on change

The equivalent update when requesting json encoding:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-change-update xmlns=
    "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>89</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-changes-json>
      {
        "ietf-yang-patch:yang-patch": {
          "patch-id": [
            null
          ],
          "edit": [
            {
              "edit-id": "edit1",
              "operation": "merge",
              "target": "/alpha/beta",
              "value": {
                "beta": 1500
              }
            }
          ]
        }
      }
    </datastore-changes-json>
  </push-change-update>
</notification>
```

Figure 3: Push example for on change with JSON

When the beta leaf is deleted, the publisher may send

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-change-update xmlns=
    "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>89</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-changes-xml>
      <alpha xmlns="http://example.com/sample-data/1.0" >
        <beta urn:ietf:params:xml:ns:netconf:base:1.0:
          operation="delete"/>
      </alpha>
    </datastore-changes-xml>
  </push-change-update>
</notification>
```

Figure 4: 2nd push example for on change update

3.7. Subscription management

A [[I-D.ietf-netconf-5277bis] subscription needs enhancement to support YANG Push subscription negotiation. Specifically, these enhancements are needed to signal to the subscriber why an attempt has failed.

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, the lack of read authorization on the requested data node, or the inability of the publisher to provide a stream with the requested semantics. In such cases, no subscription is established. Instead, the subscription-result with the failure reason is returned as part of the RPC response. In addition, a set of alternative subscription parameters MAY be returned that would likely have resulted in acceptance of the subscription request, which the subscriber may try for a future subscription attempt.

It should be noted that a rejected subscription does not result in the generation of an rpc-reply with an rpc-error element, as neither the specification of YANG-push specific errors nor the specification of additional data parameters to be returned in an error case are supported as part of a YANG data model.

For instance, for the following request:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period>500</period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 5: Establish-Subscription example

the publisher might return:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="http://urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    error-insufficient-resources
  </subscription-result>
  <period>2000</period>
</rpc-reply>
```

Figure 6: Error response example

3.8. Other considerations

3.8.1. Authorization

A receiver of subscription data may only be sent updates for which they have proper authorization. Data that is being pushed therefore needs to be subjected to a filter that applies all corresponding rules applicable at the time of a specific pushed update, silently removing any non-authorized data from subtrees.

The authorization model for data in YANG datastores is described in the Netconf Access Control Model [RFC6536]. However, some clarifications to that RFC are needed so that the desired access control behavior is applied to pushed updates.

One of these clarifications is that a subscription may only be established if the receiver has read access to the target data node.

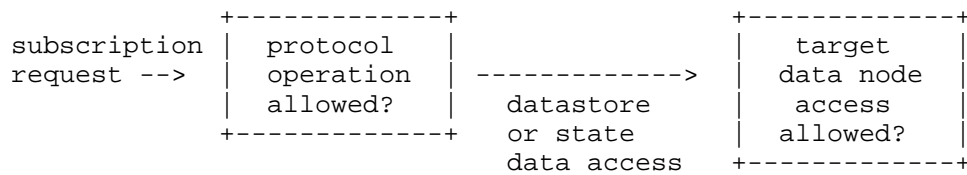


Figure 7: Access control for subscription

Likewise if a receiver no longer has read access permission to a target data node, the subscription must be abnormally terminated (with loss of access permission as the reason provided).

Another clarification to [RFC6536] is that each of the individual nodes in a pushed update must also go through access control filtering. This includes new nodes added since the last update notification, as well as existing nodes. For each of these read access must be verified. The methods of doing this efficiently are left to implementation.

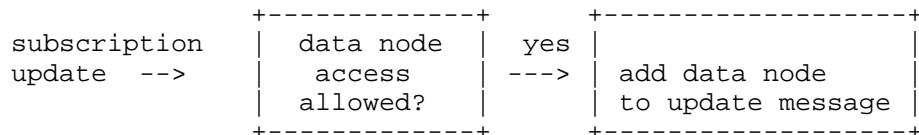


Figure 8: Access control for push updates

If there are read access control changes applied under the target node, no notifications indicating the fact that this has occurred should be provided.

3.8.2. Robustness and reliability considerations

Particularly in the case of on-change push updates, it is important that push updates do not get lost.

Update notifications will typically traverse a secure and reliable transport. Notifications will not be reordered, and will also contain a time stamp. Despite these protections for on-change, it is possible that complete update notifications get lost. For this reason, patch-ids may be included in a subscription so that an application can determine if an update has been lost.

At the same time, it is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update notification the full set of objects desired per the terms of

a subscription. In this case, the publisher must take one or more of the following actions.

- o A publisher must set the updates-not-sent flag on any update notification which is known to be missing information.
- o It may choose to suspend and resume a subscription as per [I-D.ietf-netconf-5277bis].
- o When resuming an on-change subscription, the publisher should generate a complete patch from the previous update notification. If this is not possible and the synch-on-start option is configured, then the full datastore contents may be sent instead (effectively replacing the previous contents). If neither of these are possible, then an updates-not-sent flag must be included on the next push-change-update.

3.8.3. Update size and fragmentation considerations

Depending on the subscription, the volume of updates can become quite large. There is no inherent limitation to the amount of data that can be included in a notification. That said, it may not always be practical to send the entire update in a single chunk. Implementations MAY therefore choose, at their discretion, to "chunk" updates and break them out into several update notifications.

3.8.4. Push data streams

There are several conceptual data streams introduced in this specification:

- o yang-push includes the entirety of YANG data, including both configuration and operational data.
- o operational-push includes all operational (read-only) YANG data
- o config-push includes all YANG configuration data.

It is conceivable to introduce other data streams with more limited scope, for example:

- o operdata-nocounts-push, a datastream containing all operational (read-only) data with the exception of counters
- o other custom datastreams

Those data streams make particular sense for use cases involving service assurance (not relying on operational data), and for use

cases requiring on-change update triggers which make no sense to support in conjunction with fast-changing counters. While it is possible to specify subtree filters on yang-push to the same effect, having those data streams greatly simplifies articulating subscriptions in such scenarios.

3.8.5. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, it should be noted that monitoring of operational state changes inside a system can be associated with significant implementation challenges.

Even periodic retrieval and push of operational counters may consume considerable system resources. In addition the on-change push of small amounts of configuration data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, it is important for an implementation to understand what subscriptions it can or cannot support. It is far preferable to decline a subscription request than to accept such a request when it cannot be met.

Whether or not a subscription can be supported will in general be determined by a combination of several factors, including the subscription policy (on-change or periodic, with on-change in general being the more challenging of the two), the period in which to report changes (1 second periods will consume more resources than 1 hour periods), the amount of data in the subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.

When providing access control to every node in a pushed update, it is possible to make and update efficient access control filters for an update. These filters can be set upon subscription and applied against a stream of updates. These filters need only be updated when (a) there is a new node added/removed from the subscribed tree with different permissions than its parent, or (b) read access permissions have been changed on nodes under the target node for the subscriber.

3.8.6. Not Notifiable YANG Objects

In some cases, a publisher supporting "on-change" notifications may not be able to push updates for some object types "on-change". Reasons for this might be that the value of the data node changes frequently (e.g., a received-octets-counter), that small object changes are frequent and meaningless (e.g., a temperature gauge

changing 0.1 degrees), or that the implementation is not capable of on-change notification of an object type.

The default assumption is that changes on all data nodes will be reported on-change. However if a certain data node cannot do this, it SHOULD be marked with the YANG extension not-notifiable-on-change.

4. A YANG data model for management of datastore push subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figure. Following Yang tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses with a name in the middle enclose choice and case nodes. A "+" at the end of a line indicates that the line is to be concatenated with the subsequent line. New YANG tree notation is the `i]` which indicates that the node in that line has been brought in / imported from another model, and an `(a)` which indicates this is the specific imported node augmented. In the figure below, all have been imported from 5277bis. The model consists mostly of augmentations to RPCs and notifications defined in the data model for subscriptions for event notifications of [I-D.ietf-netconf-5277bis].

```

module: ietf-yang-push
i]    +--ro streams
i]    |   +--ro stream*    stream
i]    +--rw filters
i]    |   +--rw filter* [filter-id]
i]    |   +--rw filter-id  filter-id
i]    |   +--rw (filter-type)?
i]    |   |   +--:(rfc5277)
i]    |   |   |   +--rw filter?
i]    |   |   |   +--:(update-filter)
i]    |   |   |   +--rw (update-filter)?
i]    |   |   |   |   +--:(subtree)
i]    |   |   |   |   |   +--rw subtree-filter?
i]    |   |   |   |   |   +--:(xpath)
i]    |   |   |   |   |   +--rw xpath-filter? yang:xpath1.0
i]    +--rw subscription-config {configured-subscriptions}?
i]    |   +--rw subscription* [subscription-id]
i]    |   +--rw subscription-id          subscription-id
i]    |   +--rw stream?                  stream
i]    |   +--rw encoding?                encoding
(a)   |   +--rw (filter-type)?
i]    |   |   +--:(rfc5277)

```

```

i] | | | +--rw filter?
    | | | +---:(update-filter)
    | | | | +--rw (update-filter)?
    | | | | | +---:(subtree)
    | | | | | | +--ro subtree-filter?
    | | | | | +---:(xpath)
    | | | | | +--rw xpath-filter? yang:xpath1.0
i] | | | +---:(by-reference)
i] | | | | +--rw filter-ref? filter-ref
i] | | | +--rw startTime? yang:date-and-time
i] | | | +--rw stopTime? yang:date-and-time
    | | | +--rw (update-trigger)?
    | | | | +---:(periodic)
    | | | | | +--rw period yang:timeticks
    | | | | | +--rw anchor-time? yang:date-and-time
    | | | | +---:(on-change) {on-change}?
    | | | | | +--rw no-synch-on-start? empty
    | | | | | +--rw dampening-period yang:timeticks
    | | | | | +--rw excluded-change* change-type
i] | | | +--rw receivers
i] | | | | +--rw receiver* [address]
i] | | | | | +--rw address inet:host
i] | | | | | +--rw port inet:port-number
i] | | | | | +--rw protocol? transport-protocol
i] | | | +--rw (push-source)?
i] | | | | +---:(interface-originated)
i] | | | | | +--rw source-interface? if:interface-ref
i] | | | | +---:(address-originated)
i] | | | | | +--rw source-vrf? uint32
i] | | | | | +--rw source-address inet:ip-address-no-zone
    | | | +--rw dscp? inet:dscp
    | | | +--rw subscription-priority? uint8
    | | | +--rw subscription-dependency? string
(a) +--ro subscriptions
i] | | | +--ro subscription*
i] | | | | +--ro subscription-id
i] | | | | +--ro configured-subscription?
i] | | | | +--ro subscription-status?
i] | | | | +--ro stream?
i] | | | | +--ro encoding?
(a) +--ro (filter-type)?
i] | | | | +---:(rfc5277)
i] | | | | | +--ro filter?
    | | | | +---:(update-filter)
    | | | | | +--ro (update-filter)?
    | | | | | | +---:(subtree)
    | | | | | | | +--ro subtree-filter?
    | | | | | | +---:(xpath)

```

```

| | | | | +--ro xpath-filter?
i] | | | | | +---:(by-reference)
i] | | | | | +---ro filter-ref?
i] | | | | | +---ro startTime?
i] | | | | | +---ro stopTime?
| | | | | +---ro (update-trigger)?
| | | | | | +---:(periodic)
| | | | | | | +---ro period
| | | | | | | +---ro anchor-time?
| | | | | | +---:(on-change) {on-change}?
| | | | | | | +---ro no-synch-on-start?
| | | | | | | +---ro dampening-period
| | | | | | | +---ro excluded-change*
i] | | | | | +---ro receivers
i] | | | | | | +---ro receiver*
i] | | | | | | +---ro address
i] | | | | | | +---ro port
i] | | | | | | +---ro protocol?
i] | | | | | +---ro (push-source)?
i] | | | | | | +---:(interface-originated)
i] | | | | | | | +---ro source-interface?
i] | | | | | | +---:(address-originated)
i] | | | | | | | +---ro source-vrf?
i] | | | | | | +---ro source-address
+---ro dscp?
+---ro subscription-priority?
+---ro subscription-dependency?

i] rpcs:
i] +---x establish-subscription
(a) | | | | | +---w input
i] | | | | | | +---w stream?
i] | | | | | | +---w encoding?
(a) | | | | | +---w (filter-type)?
i] | | | | | | +---:(rfc5277)
i] | | | | | | | +---w filter?
| | | | | | +---:(update-filter)
| | | | | | | +---w (update-filter)?
| | | | | | | +---:(subtree)
| | | | | | | | +---w subtree-filter?
| | | | | | | +---:(xpath)
| | | | | | | | +---w xpath-filter?
i] | | | | | +---:(by-reference)
i] | | | | | | +---w filter-ref?
i] | | | | | +---w startTime?
i] | | | | | +---w stopTime?
+---w (update-trigger)?
| | | | | +---:(periodic)
| | | | | | +---w period
```

```

| | | | +---w anchor-time?
| | | | +---:(on-change) {on-change}?
| | | | +---w no-synch-on-start?
| | | | +---w dampening-period
| | | | +---w excluded-change*
| | | +---w dscp?
| | | +---w subscription-priority?
| | | +---w subscription-dependency?
i] +--ro output
i] +--ro subscription-result
i] +--ro (result)?
i] | +---:(success)
i] | | +--ro subscription-id
(a) | +---:(no-success)
i] | +--ro stream?
i] | +--ro encoding?
(a) | +--ro (filter-type)?
i] | | +---:(rfc5277)
i] | | | +--ro filter?
| | | | +---:(update-filter)
| | | | | +---w (update-filter)?
| | | | | +---:(subtree)
| | | | | | +---w subtree-filter?
| | | | | +---:(xpath)
| | | | | | +---w xpath-filter?
i] | | +---:(by-reference)
i] | | +--ro filter-ref?
i] | +--ro startTime?
i] | +--ro stopTime?
| | | +--ro (update-trigger)?
| | | | +---:(periodic)
| | | | | +--ro period
| | | | | +--ro anchor-time?
| | | | +---:(on-change) {on-change}?
| | | | +--ro no-synch-on-start?
| | | | +--ro dampening-period
| | | | +--ro excluded-change*
| | | +--ro dscp?
| | | +--ro subscription-priority?
| | | +--ro subscription-dependency?
i] +---x modify-subscription
i] | +---w input
i] | | +---w subscription-id?
i] | | +---w (filter-type)?
i] | | | +---:(rfc5277)
i] | | | | +---w filter?
i] | | | | +---:(update-filter)
| | | | | +---w (update-filter)?

```

```

i]      +---:(subtree)
i]      |      +---w subtree-filter?
i]      |      +---:(xpath)
i]      |      +---w xpath-filter?
i]      +---:(by-reference)
i]      |      +---w filter-ref?
i]      +---w startTime?
i]      +---w stopTime?
i]      +---w (update-trigger)?
i]      |      +---:(periodic)
i]      |      |      +---w period
i]      |      |      +---w anchor-time?
i]      |      +---:(on-change) {on-change}?
i]      |      +---w dampening-period
i]      |      +---w excluded-change*
i]      +---ro output
i]      +---ro subscription-result
i]      +---ro (result)?
i]      |      +---:(success)
i]      |      |      +---ro subscription-id
i]      |      +---:(no-success)
i]      |      +---ro stream?
i]      |      +---ro encoding?
i]      |      +---ro (filter-type)?
i]      |      |      +---:(rfc5277)
i]      |      |      |      +---ro filter?
i]      |      |      +---:(update-filter)
i]      |      |      |      +---w (update-filter)?
i]      |      |      |      |      +---:(subtree)
i]      |      |      |      |      |      +---w subtree-filter?
i]      |      |      |      |      +---:(xpath)
i]      |      |      |      |      |      +---w xpath-filter?
i]      |      |      +---:(by-reference)
i]      |      |      +---ro filter-ref?
i]      |      +---ro startTime?
i]      |      +---ro stopTime?
i]      |      +---ro (update-trigger)?
i]      |      |      +---:(periodic)
i]      |      |      |      +---ro period
i]      |      |      |      +---ro anchor-time?
i]      |      |      +---:(on-change) {on-change}?
i]      |      |      +---ro no-synch-on-start?
i]      |      |      +---ro dampening-period
i]      |      |      +---ro excluded-change*
i]      |      +---ro dscp?
i]      |      +---ro subscription-priority?
i]      |      +---ro subscription-dependency?
i]      +---x delete-subscription

```

```

i]      +---w input
i]      |   +---w subscription-id
i]      +---ro output
i]      +---ro subscription-result

(a)      notifications
(a)      +---n subscription-started
i]      |   +---ro subscription-id
i]      |   +---ro stream?
i]      |   +---ro encoding?
(a)      |   +---ro (filter-type)?
i]      |   |   +---:(rfc5277)
i]      |   |   |   +---ro filter?
i]      |   |   +---:(update-filter)
i]      |   |   |   +---rw (update-filter)?
i]      |   |   |   +---:(subtree)
i]      |   |   |   |   +---ro subtree-filter?
i]      |   |   |   +---:(xpath)
i]      |   |   |   +---rw xpath-filter?
i]      |   +---:(by-reference)
i]      |   +---ro filter-ref?
i]      |   +---ro startTime?
i]      |   +---ro stopTime?
i]      |   +---ro (update-trigger)?
i]      |   |   +---:(periodic)
i]      |   |   |   +---ro period
i]      |   |   |   +---ro anchor-time?
i]      |   |   +---:(on-change) {on-change}?
i]      |   |   +---ro no-synch-on-start?
i]      |   |   +---ro dampening-period
i]      |   |   +---ro excluded-change*
i]      |   +---ro dscp?
i]      |   +---ro subscription-priority?
i]      |   +---ro subscription-dependency?
(a)      +---n subscription-modified
i]      |   +---ro subscription-id
i]      |   +---ro stream?
i]      |   +---ro encoding?
(a)      |   +---ro (filter-type)?
i]      |   |   +---:(rfc5277)
i]      |   |   |   +---ro filter?
i]      |   |   +---:(update-filter)
i]      |   |   |   +---rw (update-filter)?
i]      |   |   |   +---:(subtree)
i]      |   |   |   |   +---ro subtree-filter?
i]      |   |   |   +---:(xpath)
i]      |   |   |   +---rw xpath-filter?
i]      |   +---:(by-reference)

```

```

i] | |      +--ro filter-ref?
i] | +--ro startTime?
i] | +--ro stopTime?
   | +--ro (update-trigger)?
   | |   +--:(periodic)
   | |   |   +--ro period
   | |   |   +--ro anchor-time?
   | |   +--:(on-change) {on-change}?
   | |   +--ro no-synch-on-start?
   | |   +--ro dampening-period
   | |   +--ro excluded-change*
   | +--ro dscp?
   | +--ro subscription-priority?
   | +--ro subscription-dependency?
i] +---n subscription-terminated
i] |   +--ro subscription-id
i] |   +--ro reason?
i] +---n subscription-suspended
i] |   +--ro subscription-id
i] |   +--ro reason?
i] +---n subscription-resumed
i] |   +--ro subscription-id
i] +---n replay-complete
i] |   +--ro subscription-id
i] +---n notification-complete
i] |   +--ro subscription-id
   +---n push-update
   |   +--ro subscription-id
   |   +--ro time-of-update?
   |   +--ro updates-not-sent?
   |   +--ro datastore-contents?
   +---n push-change-update {on-change}?
       +--ro subscription-id
       +--ro time-of-update?
       +--ro updates-not-sent?
       +--ro datastore-changes?

```

Figure 9: Model structure

The components of the model are described in the following subsections.

4.2. Update streams

Container "update-streams" is used to indicate which data streams are provided by the system and can be subscribed to. For this purpose, it contains a leaf list of data nodes identifying the supported streams.

4.3. Filters

Container "filters" contains a list of configurable data filters, each specified in its own list element. This allows users to configure filters separately from an actual subscription, which can then be referenced from a subscription. This facilitates the reuse of filter definitions, which can be important in case of complex filter conditions.

One of three types of filters can be specified as part of a filter list element. Subtree filters follow syntax and semantics of RFC 6241 and allow to specify which subtree(s) to subscribe to. In addition, XPath filters can be specified for more complex filter conditions. Finally, filters can be specified using syntax and semantics of RFC5277.

It is conceivable to introduce other types of filters; in that case, the data model needs to be augmented accordingly.

4.4. Subscription configuration

As an optional feature, configured-subscriptions, allows for the configuration of subscriptions as opposed to RPC. Subscriptions configurations are represented by list subscription-config. Each subscription is represented through its own list element and includes the following components:

- o "subscription-id" is an identifier used to refer to the subscription.
- o "stream" refers to the stream being subscribed to. The subscription model assumes the presence of perpetual and continuous streams of updates. Various streams are defined: "push-update" covers the entire set of YANG data in the publisher. "operational-push" covers all operational data, while "config-push" covers all configuration data. Other streams could be introduced in augmentations to the model by introducing additional identities.
- o "encoding" refers to the encoding requested for the data updates. By default, updates are encoded using XML. However, JSON can be requested as an option if the json-encoding feature is supported. Other encodings may be supported in the future.
- o "anchor-time" is a timestamp. When used in conjunction with period, the boundaries of periodic update periods may be calculated.

- o Filters for a subscription can be specified using a choice, allowing to either reference a filter that has been separately configured or entering its definition inline.
- o A choice of subscription policies allows to define when to send new updates - periodic or on change.
 - * For periodic subscriptions, the trigger is defined by a "period", a parameter that defines the interval with which updates are to be pushed. The start time of the subscription serves as anchor time, defining one specific point in time at which an update needs to be sent. Update intervals always fall on the points in time that are a multiple of a period after the start time.
 - * For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that is guided by additional parameters. "dampening-period" specifies the interval that must pass before a successive update for the same data node is sent. The first time a change is detected, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed, containing the value of the data node that is then valid. "excluded-change" allows to restrict the types of changes for which updates are sent (changes to object values, object creation or deletion events). "no-synch-on-start" is a flag that allows to specify whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription; if the flag is omitted, a complete update is sent to facilitate synchronization. It is conceivable to augment the data model with additional parameters in the future to specify even more refined policies, such as parameters that specify the magnitude of a change that must occur before an update is triggered.
- o This is followed with a list of receivers for the subscription, indicating for each receiver the transport that should be used for push updates (if options other than Netconf are supported). It should be noted that the receiver does not have to be the same system that configures the subscription.
- o Finally, "push-source" can be used to specify the source of push updates, either a specific interface or publisher address.

A subscription established through configuration cannot be deleted using an RPC. Likewise, subscriptions established through RPC cannot be deleted through configuration.

The deletion of a subscription, whether through RPC or configuration, results in immediate termination of the subscription.

4.5. Subscription monitoring

Subscriptions can be subjected to management themselves. For example, it is possible that a publisher may no longer be able to serve a subscription that it had previously accepted. Perhaps it has run out of resources, or internal errors may have occurred. When this is the case, a publisher needs to be able to temporarily suspend the subscription, or even to terminate it. More generally, the publisher should provide a means by which the status of subscriptions can be monitored.

Container "subscriptions" contains the state of all subscriptions that are currently active. This includes subscriptions that were established (and have not yet been deleted) using RPCs, as well as subscriptions that have been configured as part of configuration.

Each subscription is represented as a list element "datastore-push-subscription". The associated information includes an identifier for the subscription, a subscription status, as well as the various subscription parameters that are in effect. The subscription status indicates whether the subscription is currently active and healthy, or if it is degraded in some form. Leaf "configured-subscription" indicates whether the subscription came into being via configuration or via RPC.

Subscriptions that were established by RPC are removed from the list once they expire (reaching stop-time)or when they are terminated. Subscriptions that were established by configuration need to be deleted from the configuration by a configuration editing operation.

4.6. Notifications

4.6.1. Monitoring and OAM Notifications

OAM notifications are reused from [I-D.ietf-netconf-5277bis]. Some have augmentations to include new objects defined in this draft.

Still to be investigated is whether a publisher might also provide additional information about subscriptions, such as statistics about the number of data updates that were sent. However, such information is currently outside the scope of this specification.

4.6.2. Update Notifications

The data model introduces two YANG notifications for the actual updates themselves.

Notification "push-update" is used to send a complete snapshot of the data that has been subscribed to, with all YANG object filters applied. The notification is used for periodic subscription updates in a periodic subscription.

The notification can also be used in an on-change subscription for the purposes of allowing a receiver to "synch". Specifically, it is used at the start of an on-change subscription, unless no-synch-on-start is specified for the subscription. In addition, it MAY be used during the subscription, for example if change updates were not sent as indicated by the "updates-not-sent" flag (see below), or for synch updates at longer period intervals (such as once per day) to mitigate the possibility of any application-dependent synchronization drift. The trigger for sending a push-update notification in conjunction with on-change subscriptions are at this point outside the scope of the specification.

The format and syntax of the contained data corresponds to the format and syntax of data that would be returned in a corresponding get operation with the same filter parameters applied.

Notification "push-change-update" is used to send data updates for changes that have occurred in the subscribed data. This notification is used only in conjunction with on-change subscriptions.

The data updates are encoded analogous to the syntax of a corresponding yang-patch operation. It corresponds to the data that would be contained in a yang-patch operation applied to the YANG datastore at the previous update, to result in the current state (and applying it also to operational data).

In rare circumstances, the notification can include a flag "updates-not-sent". This is a flag which indicates that not all changes which have occurred since the last update are actually included with this update. In other words, the publisher has failed to fulfill its full subscription obligations, for example in cases where it was not able to keep up with a change burst. To facilitate synchronization, a publisher MAY subsequently send a push-update containing a full snapshot of subscribed data. Such a push-update might also be triggered by a subscriber requesting an on-demand synchronization.

4.7. RPCs

YANG-Push subscriptions are established, modified, and deleted using three RPCs.

4.7.1. Establish-subscription RPC

The subscriber sends an establish-subscription RPC with the parameters in section 3.1. For instance

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period>500</period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 10: Establish-subscription RPC

The publisher must respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a publisher may respond:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    ok
  </subscription-result>
  <subscription-id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    52
  </subscription-id>
</rpc-reply>
```

Figure 11: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, the lack of read authorization on the requested data node, or the inability of the publisher to provide a stream with the requested semantics.

When the requester is not authorized to read the requested data node, the returned "error-info"; indicates an authorization error and the requested node. For instance, if the above request was unauthorized to read node "ex:foo" the publisher may return:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    error-data-not-authorized
  </subscription-result>
</rpc-reply>
```

Figure 12: Establish-subscription access denied response

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error what subscription parameters would have been accepted for the request. However, there are no guarantee that subsequent requests for this subscriber or others will in fact be accepted.

For example, for the following request:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <dampening-period>10</dampening-period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 13: Establish-subscription request example 2

A publisher that cannot serve on-change updates but periodic updates might return the following:

```

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    error-no-such-option
  </subscription-result>
  <period>100</period>
</rpc-reply>

```

Figure 14: Establish-subscription error response example 2

4.7.2. Modify-subscription RPC

The subscriber may send a modify-subscription RPC for a subscription previously established using RPC. The subscriber may change any subscription parameters by including the new values in the modify-subscription RPC. Parameters not included in the rpc should remain unmodified. For illustration purposes we include an exchange example where a subscriber modifies the period of the subscription.

```

<netconf:rpc message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <stream>push-update</stream>
    <subscription-id>
      1011
    </subscription-id>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period>250</period>
    <encoding>encode-xml</encoding>
  </modify-subscription>
</netconf:rpc>

```

Figure 15: Modify subscription request

The publisher must respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a publisher may respond:

```

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    ok
  </subscription-result>
  <subscription-id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    1011
  </subscription-id>
</rpc-reply>

```

Figure 16: Modify subscription response

If the subscription modification is rejected, the publisher must send a response like it does for an establish-subscription and maintain the subscription as it was before the modification request. A subscription may be modified multiple times.

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

4.7.3. Delete-subscription RPC

To stop receiving updates from a subscription and effectively delete a subscription that had previously been established using an establish-subscription RPC, a subscriber can send a delete-subscription RPC, which takes as only input the subscription-id. For example:

```

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

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

```

Figure 17: Delete subscription

Configured subscriptions cannot be deleted via RPC, but have to be removed from the configuration.

4.7.4. YANG Module Synchronization

In order to fully support datastore replication, the receiver needs to know the YANG module library that is in use by server that is being replicated. The YANG 1.0 module library information is sent by a NETCONF server in the NETCONF 'hello' message. For YANG 1.1 modules and all modules used with the RESTCONF [I-D.ietf-netconf-restconf] protocol, this information is provided by the YANG Library module (ietf-yang-library.yang from [RFC7895]). The YANG library information is important for the receiver to reproduce the set of object definitions used by the replicated datastore.

The YANG library includes a module list with the name, revision, enabled features, and applied deviations for each YANG module implemented by the server. The receiver is expected to know the YANG library information before starting a subscription. The "/modules-state/module-set-id" leaf in the "ietf-yang-library" module can be used to cache the YANG library information. [ED. NOTE: Should "module-set-id" be added to establish-subscription response?]

The set of modules, revisions, features, and deviations can change at run-time (if supported by the server implementation). In this case, the receiver needs to be informed of module changes before data nodes from changed modules can be processed correctly. The YANG library provides a simple "yang-library-change" notification that informs the client that the library has changed somehow. The receiver then needs to re-read the entire YANG library data for the replicated server in order to detect the specific YANG library changes. The "ietf-netconf-notifications" module defined in [RFC6470] contains a "netconf-capability-change" notification that can identify specific module changes. For example, the module URI capability of a newly loaded module will be listed in the "added-capability" leaf-list, and the module URI capability of an removed module will be listed in the "deleted-capability" leaf-list.

5. YANG module

```
<CODE BEGINS> file "ietf-yang-push@2016-10-28.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-inet-types {
    prefix inet;
```



```
}
import ietf-yang-types {
  prefix yang;
}
import ietf-event-notifications {
  prefix notif-bis;
}

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

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

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

  Editor:    Alexander Clemm
             <mailto:alex@sympotech.com>

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

  Editor:    Alberto Gonzalez Prieto
             <mailto:albertgo@cisco.com>

  Editor:    Ambika Prasad Tripathy
             <mailto:ambtripa@cisco.com>

  Editor:    Einar Nilsen-Nygaard
             <mailto:einarnn@cisco.com>

  Editor:    Andy Bierman
             <mailto:andy@yumaworks.com>

  Editor:    Balazs Lengyel
             <mailto:balazs.lengyel@ericsson.com>";

description
  "This module contains conceptual YANG specifications
  for YANG push.";

revision 2016-10-28 {
  description
    "Updates to simplify modify-subscription, add anchor-time";
  reference "YANG Datastore Push, draft-ietf-netconf-yang-push-04";
```

```
    }

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

/*
 * IDENTITIES
 */

/* Additional errors for subscription operations */
identity error-data-not-authorized {
  base notif-bis:error;
  description
    "No read authorization for a requested data node.";
}

/* Additional types of streams */
identity update-stream {
  description
    "Base identity to represent a conceptual system-provided
    datastream of datastore updates with predefined semantics.";
}

identity yang-push {
  base update-stream;
  description
    "A conceptual datastream consisting of all datastore
    updates, including operational and configuration data.";
}

identity operational-push {
  base update-stream;
  description
    "A conceptual datastream consisting of updates of all
    operational data.";
}

identity config-push {
  base update-stream;
  description
    "A conceptual datastream consisting of updates of all
    configuration data.";
}
```

```
identity custom-stream {
  base update-stream;
  description
    "A conceptual datastream for datastore
    updates with custom updates as defined by a user.";
}

/* Additional transport option */
identity http2 {
  base notif-bis:transport;
  description
    "HTTP2 notifications as a transport";
}

/*
 * TYPE DEFINITIONS
 */

typedef filter-id {
  type uint32;
  description
    "A type to identify filters which can be associated with a
    subscription.";
}

typedef change-type {
  type enumeration {
    enum "create" {
      description
        "A new data node was created";
    }
    enum "delete" {
      description
        "A data node was deleted";
    }
    enum "modify" {
      description
        "The value of a data node has changed";
    }
  }
  description
    "Specifies different types of changes that may occur
    to a datastore.";
}

typedef update-stream {
  type identityref {
    base update-stream;
  }
}
```

```
    }
    description
      "Specifies a system-provided datastream.";
  }

  grouping update-filter {
    description
      "This groupings defines filters for push updates for a
      datastore tree. The filters define which updates are of
      interest in a push update subscription. Mixing and matching
      of multiple filters does not occur at the level of this
      grouping. When a push-update subscription is created, the
      filter can be a regular subscription filter, or one of the
      additional filters that are defined in this grouping.";
    choice update-filter {
      description
        "Define filters regarding which data nodes to include
        in push updates";
      case subtree {
        description
          "Subtree filter.";
        anyxml subtree-filter {
          description
            "Subtree-filter used to specify the data nodes targeted
            for subscription within a subtree, or subtrees, of a
            conceptual YANG datastore. Objects matching the filter
            criteria will traverse the filter. The syntax follows
            the subtree filter syntax specified in RFC 6241,
            section 6.";
          reference "RFC 6241 section 6";
        }
      }
      case xpath {
        description
          "XPath filter";
        leaf xpath-filter {
          type yang:xpath1.0;
          description
            "XPath defining the data items of interest.";
        }
      }
    }
  }

  grouping update-policy {
    description
      "This grouping describes the conditions under which an
      update will be sent as part of an update stream.";
```

```
choice update-trigger {
  description
    "Defines necessary conditions for sending an event to
    the subscriber.";
  case periodic {
    description
      "The agent is requested to notify periodically the
      current values of the datastore or the subset
      defined by the filter.";
    leaf period {
      type yang:timeticks;
      mandatory true;
      description
        "Duration of time which should occur between periodic
        push updates. Where the anchor of a start-time is
        available, the push will include the objects and their
        values which exist at an exact multiple of timeticks
        aligning to this start-time anchor.";
    }
    leaf anchor-time {
      type yang:date-and-time;
      description
        "Designates a timestamp from which the series of
        periodic push updates are computed. The next update
        will take place at the next period interval from the
        anchor time. For example, for an anchor time at the
        top of a minute and a period interval of a minute,
        the next update will be sent at the top of the next
        minute.";
    }
  }
}
case on-change {
  if-feature "on-change";
  description
    "The agent is requested to notify changes in
    values in the datastore or a subset of it defined
    by a filter.";
  leaf no-synch-on-start {
    type empty;
    description
      "This leaf acts as a flag that determines behavior at the
      start of the subscription. When present,
      synchronization of state at the beginning of the
      subscription is outside the scope of the subscription.
      Only updates about changes that are observed from the
      start time, i.e. only push-change-update notifications
      are sent.
      When absent (default behavior), in order to facilitate
```

```

        a receiver's synchronization, a full update is sent
        when the subscription starts using a push-update
        notification, just like in the case of a periodic
        subscription. After that, push-change-update
        notifications only are sent unless the Publisher chooses
        to resynch the subscription again.";
    }
    leaf dampening-period {
        type yang:timeticks;
        mandatory true;
        description
            "Minimum amount of time that needs to have
            passed since the last time an update was
            provided.";
    }
    leaf-list excluded-change {
        type change-type;
        description
            "Use to restrict which changes trigger an update.
            For example, if modify is excluded, only creation and
            deletion of objects is reported.";
    }
}
}
}

grouping subscription-qos {
    description
        "This grouping describes Quality of Service information
        concerning a subscription. This information is passed to lower
        layers for transport prioritization and treatment";
    leaf dscp {
        if-feature "notif-bis:configured-subscriptions";
        type inet:dscp;
        default "0";
        description
            "The push update's IP packet transport priority.
            This is made visible across network hops to receiver.
            The transport priority is shared for all receivers of
            a given subscription.";
    }
    leaf subscription-priority {
        type uint8;
        description
            "Relative priority for a subscription. Allows an
            underlying transport layer perform informed load
            balance allocations between various subscriptions";
    }
}

```

```
leaf subscription-dependency {
  type string;
  description
    "Provides the Subscription ID of a parent subscription
    without which this subscription should not exist. In
    other words, there is no reason to stream these objects
    if another subscription is missing.";
}

augment "/notif-bis:establish-subscription/notif-bis:input" {
  description
    "Define additional subscription parameters that apply
    specifically to push updates";
  uses update-policy;
  uses subscription-qos;
}
augment "/notif-bis:establish-subscription/notif-bis:input/" +
  "notif-bis:filter-type" {
  description
    "Add push filters to selection of filter types.";
  case update-filter {
    description
      "Additional filter options for push subscription.";
    uses update-filter;
  }
}
augment "/notif-bis:establish-subscription/notif-bis:output" {
  description
    "Allow to return additional subscription parameters that apply
    specifically to push updates.";
  uses update-policy;
  uses subscription-qos;
}
augment "/notif-bis:establish-subscription/notif-bis:output/" +
  "notif-bis:result/notif-bis:no-success/notif-bis:filter-type" {
  description
    "Add push filters to selection of filter types.";
  case update-filter {
    description
      "Additional filter options for push subscription.";
    uses update-filter;
  }
}
augment "/notif-bis:modify-subscription/notif-bis:input" {
  description
    "Define additional subscription parameters that apply
    specifically to push updates.";
```

```
    uses update-policy;
  }
  augment "/notif-bis:modify-subscription/notif-bis:input/" +
    "notif-bis:filter-type" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
  augment "/notif-bis:modify-subscription/notif-bis:output" {
    description
      "Allow to return additional subscription parameters that apply
      specifically to push updates.";
    uses update-policy;
    uses subscription-qos;
  }
  augment "/notif-bis:modify-subscription/notif-bis:output/" +
    "notif-bis:result/notif-bis:no-success/notif-bis:filter-type" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
}
notification push-update {
  description
    "This notification contains a push update, containing
    data subscribed to via a subscription.
    This notification is sent for periodic updates, for a
    periodic subscription. It can also be used for
    synchronization updates of an on-change subscription.
    This notification shall only be sent to receivers
    of a subscription; it does not constitute a general-purpose
    notification.";
  leaf subscription-id {
    type notif-bis:subscription-id;
    mandatory true;
    description
      "This references the subscription because of which the
      notification is sent.";
  }
  leaf time-of-update {
    type yang:date-and-time;
  }
}
```



```
    description
      "This leaf contains the time of the update.";
  }
  leaf updates-not-sent {
    type empty;
    description
      "This is a flag which indicates that not all data nodes
      subscribed to are included included with this
      update.  In other words, the publisher has failed to
      fulfill its full subscription obligations.
      This may lead to intermittent loss of synchronization
      of data at the client.  Synchronization at the client
      can occur when the next push-update is received.";
  }
  anydata datastore-contents {
    description
      "This contains the updated data.  It constitutes a snapshot
      at the time-of-update of the set of data that has been
      subscribed to.  The format and syntax of the data
      corresponds to the format and syntax of data that would be
      returned in a corresponding get operation with the same
      filter parameters applied.";
  }
}
notification push-change-update {
  if-feature "on-change";
  description
    "This notification contains an on-change push update.
    This notification shall only be sent to the receivers
    of a subscription; it does not constitute a general-purpose
    notification.";
  leaf subscription-id {
    type notif-bis:subscription-id;
    mandatory true;
    description
      "This references the subscription because of which the
      notification is sent.";
  }
  leaf time-of-update {
    type yang:date-and-time;
    description
      "This leaf contains the time of the update, i.e. the
      time at which the change was observed.";
  }
  leaf updates-not-sent {
    type empty;
    description
      "This is a flag which indicates that not all changes which
```

```
        have occurred since the last update are included with this
        update.  In other words, the publisher has failed to
        fulfill its full subscription obligations, for example in
        cases where it was not able to keep up with a change burst.
        To facilitate synchronization, a publisher MAY subsequently
        send a push-update containing a full snapshot of subscribed
        data. Such a push-update might also be triggered by a
        subscriber requesting an on-demand synchronization.";
    }
    anydata datastore-changes {
        description
            "This contains datastore contents that has changed
            since the previous update, per the terms of the
            subscription.  Changes are encoded analogous to
            the syntax of a corresponding yang-patch operation,
            i.e. a yang-patch operation applied to the YANG datastore
            implied by the previous update to result in the current
            state (and assuming yang-patch could also be applied to
            operational data).";
    }
}
augment "/notif-bis:subscription-started" {
    description
        "This augmentation adds push subscription parameters
        to the notification that a subscription has
        started and data updates are beginning to be sent.
        This notification shall only be sent to receivers
        of a subscription; it does not constitute a general-purpose
        notification.";
    uses update-policy;
    uses subscription-qos;
}
augment "/notif-bis:subscription-started/notif-bis:filter-type" {
    description
        "This augmentation allows to include additional update filters
        options to be included as part of the notification that a
        subscription has started.";
    case update-filter {
        description
            "Additional filter options for push subscription.";
        uses update-filter;
    }
}
}
augment "/notif-bis:subscription-modified" {
    description
        "This augmentation adds push subscription parameters
        to the notification that a subscription has
        been modified.
```

```
        This notification shall only be sent to receivers
        of a subscription; it does not constitute a general-purpose
        notification.";
    uses update-policy;
    uses subscription-qos;
}
augment "/notif-bis:subscription-modified/notif-bis:filter-type" {
    description
        "This augmentation allows to include additional update
        filters options to be included as part of the notification
        that a subscription has been modified.";
    case update-filter {
        description
            "Additional filter options for push subscription.";
        uses update-filter;
    }
}
augment "/notif-bis:filters/notif-bis:filter/" +
    "notif-bis:filter-type" {
    description
        "This container adds additional update filter options
        to the list of configurable filters
        that can be applied to subscriptions. This facilitates
        the reuse of complex filters once defined.";
    case update-filter {
        uses update-filter;
    }
}
augment "/notif-bis:subscription-config/notif-bis:subscription" {
    description
        "Contains the list of subscriptions that are configured,
        as opposed to established via RPC or other means.";
    uses update-policy;
    uses subscription-qos;
}
augment "/notif-bis:subscription-config/notif-bis:subscription/" +
    "notif-bis:filter-type" {
    description
        "Add push filters to selection of filter types.";
    case update-filter {
        uses update-filter;
    }
}
augment "/notif-bis:subscriptions/notif-bis:subscription" {
    description
        "Contains the list of currently active subscriptions,
        i.e. subscriptions that are currently in effect,
        used for subscription management and monitoring purposes.
```

```
        This includes subscriptions that have been setup via RPC
        primitives, e.g. establish-subscription, delete-subscription,
        and modify-subscription, as well as subscriptions that
        have been established via configuration.";
    uses update-policy;
    uses subscription-qos;
}
augment "/notif-bis:subscriptions/notif-bis:subscription/" +
    "notif-bis:filter-type" {
    description
        "Add push filters to selection of filter types.";
    case update-filter {
        description
            "Additional filter options for push subscription.";
        uses update-filter;
    }
}
```

<CODE ENDS>

6. Security Considerations

Subscriptions could be used to attempt to overload publishers of YANG datastores. For this reason, it is important that the publisher has the ability to decline a subscription request if it would deplete its resources. In addition, a publisher needs to be able to suspend an existing subscription when needed. When this occur, the subscription status is updated accordingly and the receivers are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a receiver has no authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

A subscription could be configured on another receiver's behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver which doesn't even support subscriptions. Receivers which do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the Netconf Authorization Control Model SHOULD be used to control and restrict authorization of subscription configuration.

7. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Sharon Chisholm, and Guangying Zheng.

8. References

8.1. Normative References

- [I-D.ietf-netconf-5277bis]
Clemm, A., Gonzalez Prieto, A., Voit, E., Tripathy, A., Nilsen-Nygaard, E., Chisholm, S., and H. Trevino, "Subscribing to YANG-Defined Event Notifications", draft-ietf-netconf-5277bis-01 (work in progress), October 2016.
- [I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-12 (work in progress), September 2016.
- [RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, February 2012.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, June 2016.
- [RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, August 2016.

8.2. Informative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-17, September 2016.
- [RFC1157] Case, J., "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, June 2016.

Appendix A. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

A.1. Unresolved and yet-to-be addressed issues

Which stream types to introduce, if any based on implications of opstate. Current list includes streams for all operational and for all config data. Consider adding stream for operational data minus counters.

We need a new Metadata filter. But so does traditional GET. This should be relevant independent of subscriptions. This has implications of ephemeral requirements from I2RS

Should we allow an interplay of filter types in a single subscription. Or should we keep them fully independent.

Do we add a counter for the number of object changes during a dampening period?

A.2. Agreement in principal

Do we need an extension for NACM to support filter out datastore nodes for which the receiver has no read access? (And how does this differ from existing GET, which must do the same filtering?) In 5277, such filtering is done at the notification level. Yang-push includes notification-content filtering. This may be very expensive in terms of processing. Andy suggestion: only accept Yang-push subscriptions for subtrees the user has rights for all the nodes in the subtree. Changes to those rights trigger a subscription termination. Should we codify this, or let vendors determine when per subtree filtering might be applied?

Need to add a new RPC to request enabling a resynch for an existing on-change subscription exposed on publisher

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v03 to v04

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

Authors' Addresses

Alexander Clemm
Sympotech

Email: alex@sympotech.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Balazs Lengyel
Ericsson

Email: balazs.lengyel@ericsson.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

K. Watsen
Juniper Networks
M. Abrahamsson
T-Systems
October 31, 2016

Zero Touch Provisioning for NETCONF or RESTCONF based Management
draft-ietf-netconf-zerotouch-11

Abstract

This draft presents a secure technique for establishing a NETCONF or RESTCONF connection between a newly deployed device, configured with just its factory default settings, and its deployment specific network management system (NMS).

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:

- o draft-ietf-netconf-call-home
- o draft-ietf-netconf-restconf
- o draft-ietf-netconf-server-model
- o draft-ietf-anima-bootstrapping-keyinfra

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

- o "XXXX" --> the assigned RFC value for this draft

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

- o "2016-10-31" --> the publication date of this draft

The following one Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Use Cases	4
1.2. Terminology	5
1.3. Requirements Language	6
1.4. Tree Diagram Notation	7
2. Guiding Principles	7
2.1. Trust Anchors	7
2.2. Conveying Trust	7
2.3. Conveying Ownership	8

3.	Types of Information	9
3.1.	Redirect Information	9
3.2.	Bootstrap Information	10
4.	Artifacts	11
4.1.	Information Type	11
4.2.	Signature	11
4.3.	Ownership Voucher	11
4.4.	Owner Certificate	12
4.5.	Voucher Revocation	12
4.6.	Certificate Revocation	13
5.	Artifact Groupings	14
5.1.	Unsigned Information	15
5.2.	Signed Information (without Revocations)	15
5.3.	Signed Information (with Revocations)	16
6.	Sources of Bootstrapping Data	16
6.1.	Removable Storage	16
6.2.	DNS Server	18
6.3.	DHCP Server	19
6.4.	Bootstrap Server	20
7.	Workflow Overview	22
7.1.	Onboarding and Ordering Devices	22
7.2.	Owner Stages the Network for Bootstrap	25
7.3.	Device Powers On	27
8.	Device Details	29
8.1.	Factory Default State	29
8.2.	Boot Sequence	30
8.3.	Processing a Source of Bootstrapping Data	31
8.4.	Validating Signed Data	32
8.5.	Processing Redirect Information	33
8.6.	Processing Bootstrap Information	34
9.	RESTCONF API for Bootstrap Servers	35
9.1.	Tree Diagram	35
9.2.	YANG Module	37
10.	Security Considerations	49
10.1.	Immutable storage for trust anchors	50
10.2.	Clock Sensitivity	50
10.3.	Blindly authenticating a bootstrap server	50
10.4.	Entropy loss over time	51
10.5.	Serial Numbers	51
10.6.	Sequencing Sources of Bootstrapping Data	51
11.	IANA Considerations	51
11.1.	The BOOTP Manufacturer Extensions and DHCP Options Registry	51
11.1.1.	DHCP v4 Option	51
11.1.2.	DHCP v6 Option	52
11.2.	The IETF XML Registry	53
11.3.	The YANG Module Names Registry	53
12.	Other Considerations	53

13. Acknowledgements	53
14. References	53
14.1. Normative References	54
14.2. Informative References	55
Appendix A. API Examples	57
A.1. Unsigned Redirect Information	57
A.2. Signed Redirect Information	58
A.3. Unsigned Bootstrap Information	61
A.4. Signed Bootstrap Information	63
A.5. Progress Notifications	67
Appendix B. Artifact Examples	69
B.1. Redirect Information	69
B.2. Bootstrap Information	69
Appendix C. Change Log	69
C.1. ID to 00	69
C.2. 00 to 01	70
C.3. 01 to 02	70
C.4. 02 to 03	71
C.5. 03 to 04	71
C.6. 04 to 05	71
C.7. 05 to 06	71
C.8. 06 to 07	72
C.9. 07 to 08	72
C.10. 08 to 09	72
C.11. 09 to 10	72
C.12. 10 to 11	73
Authors' Addresses	73

1. Introduction

A fundamental business requirement for any network operator is to reduce costs where possible. For network operators, deploying devices to many locations can be a significant cost, as sending trained specialists to each site to do installations is both cost prohibitive and does not scale.

This document defines a bootstrapping strategy enabling devices to securely obtain bootstrapping data with no installer input, beyond physical placement and connecting network and power cables. The ultimate goal of this document is to enable a secure NETCONF [RFC6241] or RESTCONF [draft-ietf-netconf-restconf] connection to the deployment specific network management system (NMS).

1.1. Use Cases

- o Connecting to a remotely administered network

This use-case involves scenarios, such as a remote branch office or convenience store, whereby a device connects as an access gateway to an ISP's network. Assuming it is not possible to customize the ISP's network to provide any bootstrapping support, and with no other nearby device to leverage, the device has no recourse but to reach out to an Internet-based bootstrap server to bootstrap off of.

- o Connecting to a locally administered network

This use-case covers all other scenarios and differs only in that the device may additionally leverage nearby devices, which may direct it to use a local service to bootstrap off of. If no such information is available, or the device is unable to use the information provided, it can then reach out to network just as it would for the remotely administered network use-case.

1.2. Terminology

This document uses the following terms:

Artifact: The term "artifact" is used throughout to represent the any of the six artifacts defined in Section 4. These artifacts collectively provide all the bootstrapping data a device needs.

Bootstrapping Data: The term "bootstrapping data" is used throughout this document to refer to the collection of data that a device may obtain from any source of bootstrapping data. Specifically, it refers to the artifacts defined in Section 4.

Bootstrap Information: The term "bootstrap information" is used herein to refer to one of the bootstrapping artifacts defined in Section 4. Specifically, bootstrap information is the bootstrapping data that guides a device to, for instance, install a specific boot-image and commit a specific configuration.

Bootstrap Server: The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 9.2.

Device: The term "device" is used throughout this document to refer to the network element that needs to be bootstrapped. See Section 8 for more information about devices.

Initial Secure Device Identifier (IDevID): The term "IDevID" is defined in [Std-802.1AR-2009] as the secure device identifier (DevID) installed on the device by the manufacturer. This

identifier is used in this document to enable a Bootstrap Server to securely identify and authenticate a device.

Manufacturer: The term "manufacturer" is used herein to refer to the manufacturer of a device or a delegate of the manufacturer.

Network Management System (NMS): The acronym "NMS" is used throughout this document to refer to the deployment specific management system that the bootstrapping process is responsible for introducing devices to. From a device's perspective, when the bootstrapping process has completed, the NMS is a NETCONF or RESTCONF client.

Owner: See Rightful Owner.

Redirect Information: The term "bootstrap information" is used herein to refer to one of the bootstrapping artifacts defined in Section 4. Specifically, redirect information is the bootstrapping data that directs a device to connect to a bootstrap server.

Redirect Server: The term "redirect server" is used to refer to a subset of bootstrap servers that only returns redirect information. A redirect server is particularly useful when hosted by a manufacturer, to redirect devices to deployment-specific bootstrap servers.

Rightful Owner: The term "rightful owner" is used herein to refer to the person or organization that purchased or otherwise owns a device. Ownership is further described in Section 2.3.

Signed Data: The term "signed data" is used throughout to mean either redirect information or bootstrap information that has been signed by a device's rightful owner's private key.

Unsigned Data: The term "unsigned data" is used throughout to mean either redirect information or bootstrap information that has not been signed by a device's rightful owner's private key.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the sections below are to be interpreted as described in RFC 2119 [RFC2119].

1.4. Tree Diagram Notation

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

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

2. Guiding Principles

This section provides overarching principles guiding the solution presented in this document.

2.1. Trust Anchors

A trust anchor is used in cryptography to represent an entity in which trust is implicit and not derived. In public key infrastructure using X.509 certificates, a root certificate is the trust anchor, from which a chain of trust is derived. The solution presented in this document requires that all the entities involved (e.g., devices, bootstrap servers, NMSs) possess specific trust anchors in order to ensure mutual authentication throughout the zero touch bootstrapping process.

2.2. Conveying Trust

A device in its factory default state possesses a limited set of manufacturer specified trust anchors. In this document, there are two types of trust anchors of interest. The first type of trust anchor is used to authenticate a secure (e.g., HTTPS) connection to, for instance, a manufacturer-hosted Internet-based bootstrap server. The second type of trust anchor is used to authenticate manufacturer-

signed data, such as the ownership voucher artifact described in Section 4.3.

Using the first type of trust anchor, trust is conveyed by the device first authenticating the server (e.g., a bootstrap server), and then by the device trusting that the server would only provide data that its rightful owner staged for it to find. Thereby the device can trust any information returned from the server.

Using the second type of trust anchor, trust is conveyed by the device first authenticating that an artifact has been signed by its rightful owner, and thereby can trust any information held within the artifact.

Notably, redirect information, as described in Section 3.1, may include more trust anchors, which illustrates another way in which trust can be conveyed.

2.3. Conveying Ownership

The ultimate goal of this document is to enable a device to establish a secure connection with its rightful owner's NMS. This entails the manufacturer being able to track who is the rightful owner of a device (not defined in this document), as well as an ability to convey that information to devices (defined in this document).

Matching the two ways to convey trust (Section 2.2), this document provides two ways to convey ownership, by using a trusted bootstrap server (Section 6.4) or by using an ownership voucher (Section 4.3).

When a device connects to a trusted bootstrap server, one that was preconfigured into its factory default configuration, it implicitly trusts that the bootstrap server would only provide data that its rightful owner staged for it to find. That is, ownership is conveyed by the administrator of the bootstrap server (e.g., a manufacturer) taking the onus of ensuring that only data configured by a device's rightful owner is made available to the device. With this approach, the assignment of a device to an owner is ephemeral, as the administrator can reassign a device to another owner at any time.

When a device is presented signed bootstrapping data, it can authenticate that its rightful owner provided the data by verifying the signature over the data using an additional artifact defined within this document, the ownership voucher. With this approach, ownership is conveyed by the manufacturer (or delegate) taking the onus of ensuring that the ownership vouchers it issues are accurate and, in some cases, also ensuring timely voucher revocations (Section 4.5).

3. Types of Information

This document defines two types of information, redirect information and bootstrap information, that devices access during the bootstrapping process. These two types of information are described in this section.

3.1. Redirect Information

Redirect information provides information to redirect a device to a bootstrap server. Redirect information encodes a list of bootstrap servers, each defined by its hostname or IP address, an optional port, and an optional trust anchor certificate.

Redirect information is YANG modeled data formally defined by the "redirect-information" grouping in the YANG module presented in Section 9.2. This grouping has the tree diagram shown below. Please see Section 1.4 for tree diagram notation.

```
+--:(redirect-information)
  +--ro redirect-information
    +--ro bootstrap-server* [address]
      +--ro address          inet:host
      +--ro port?            inet:port-number
      +--ro trust-anchor?    binary
```

Redirect information MAY be trusted or untrusted. The redirect information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's rightful owner. In all other cases, the redirect information is untrusted.

Trusted redirect information is useful for enabling a device to establish a secure connection to a bootstrap server, which is possible when the redirect information includes the bootstrap server's trust anchor certificate. When a device is able to establish a secure connection to a bootstrap server, the bootstrapping data does not have to be signed in order to be trusted, as described in Section 2.2.

Untrusted redirect information is useful for directing a device to a bootstrap server where signed data has been staged for it to obtain. When the redirect information is untrusted, the device MUST discard any potentially included trust anchor certificates. When the redirect information is untrusted, a device MAY establish a provisional connection to any of the specified bootstrap servers. A provisional connection is accomplished by the device blindly accepting the bootstrap server's TLS certificate. In this case, the

device MUST NOT trust the bootstrap server, and data provided by the bootstrap server MUST be signed for it to be of any use to the device.

How devices process redirect information is described more formally in Section 8.5.

3.2. Bootstrap Information

Bootstrap information provides all the data necessary for a device to bootstrap itself, in order to be considered ready to be managed (e.g., by an NMS). As defined in this document, this data includes information about a boot image the device MUST be running, an initial configuration the device MUST commit, and optional scripts that, if specified, the device MUST successfully execute.

Bootstrap information is YANG modeled data formally defined by the "bootstrap-information" grouping in the YANG module presented in Section 9.2. This grouping has the tree diagram shown below. Please see Section 1.4 for tree diagram notation.

```

+--:(bootstrap-information)
  +--ro bootstrap-information
    +--ro boot-image
      |   +--ro name          string
      |   +--ro (hash-algorithm)
      |   |   +--:(sha256)
      |   |   |   +--ro sha256?    string
      |   |   +--ro uri*         inet:uri
      +--ro configuration-handling    enumeration
      +--ro pre-configuration-script?  script
      +--ro configuration?
      +--ro post-configuration-script?  script

```

Bootstrap information MUST be trusted for it to be of any use to a device. There is no option for a device to process untrusted bootstrap information.

Bootstrap information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's rightful owner. In all other cases, the bootstrap information is untrusted.

How devices process bootstrap information is described more formally in Section 8.6.

4. Artifacts

This document defines six artifacts that can be made available to devices while they are bootstrapping. As will be seen in Section 6, each source of bootstrapping information specifies a means for providing each of the artifacts defined in this section.

4.1. Information Type

The information type artifact encodes the essential bootstrapping data for the device. This artifact is used to encode the redirect information and bootstrap information types discussed in Section 3.

The information type artifact is YANG modeled data formally defined by the "information-type" choice node in Section 9.2 and can be encoded using any standard YANG encoding (e.g., XML, JSON).

4.2. Signature

The signature artifact is used by a device to verify that an information type artifact was created by the device's rightful owner. The signature is generated using the owner's private key over the information-type artifact, in whatever encoding it is presented in (e.g., XML, JSON, etc.). How signed data is validated is formally described in Section 8.4.

The signature artifact is formally a PKCS#7 SignedData structure as specified by Section 9.1 of [RFC2315], containing just the signature (no content, certificates, or CRLs), encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

4.3. Ownership Voucher

The ownership voucher is used to securely identify a device's owner, as it is known to the manufacturer. The ownership voucher is signed by the device's manufacturer or delegate.

The ownership voucher is used by a device to verify the owner certificate (Section 4.4) that the device SHOULD have also received, as described in Section 5. In particular, the device verifies that owner certificate's chain of trust includes the trusted certificate included in the voucher, and also verifies that the owner certificate contains an identifier matching the one specified in the voucher.

In order to validate the voucher, a device MUST verify that the voucher was signed by the private key associated with a trusted certificate known to the device in its factory default state, as described in Section 8.1, and the device MUST verify that the

voucher's expression for the devices that it applies to includes the device's unique identifier (e.g., serial number) and, for devices that insist on verifying voucher revocation status, the device **MUST** verify that the voucher has neither expired nor been revoked.

The ownership voucher artifact, including its encoding, is formally defined in [draft-kwatsen-netconf-voucher].

4.4. Owner Certificate

The owner certificate artifact is a certificate that is used to identify an 'owner' (e.g., an organization), as known to a trusted certificate authority. The owner certificate is signed by the trusted certificate authority.

The owner certificate is used by a device to verify the signature artifact (Section 4.2) that the device **SHOULD** have also received, as described in Section 5. In particular, the device verifies signature using the public key in the owner certificate over the information type artifact (Section 4.1).

In order to validate the owner certificate, a device **MUST** verify that the owner certificate's certificate chain includes the certificate specified by the ownership voucher (Section 4.3) that the device **SHOULD** have also received, as described in Section 5, and the device **MUST** verify that owner certificate contains an identifier matching the one specified in the voucher and, for devices that insist on verifying certificate revocation status, the device **MUST** verify that the certificate has neither expired nor been revoked.

The owner certificate artifact is formally an unsigned PKCS #7 SignedData structure as specified by RFC 2315 [RFC2315], Section 9.1, containing just certificates (no content, signatures, or CRLs), encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

The owner certificate artifact contains, in order, the owner certificate itself and all intermediate certificates leading up to a trust anchor certificate. The owner certificate **MAY** optionally include the trust anchor certificate.

4.5. Voucher Revocation

The voucher revocation artifact is used to verify the revocation status of vouchers. Voucher revocations are signed by the manufacturer or delegate (i.e. the issuer of the voucher).

Voucher revocations are generally needed when it is critical for devices to know that assurances implied at the time the voucher was signed are still valid at the time the voucher is being processed.

The need for devices to insist on verifying voucher revocation status is a decision for each manufacturer. If voucher revocation status verification is not asserted, then the ownership vouchers are essentially forever, which may be acceptable for various kinds of devices. If revocations are supported, then it becomes possible to support various scenarios such as handling a key compromise or change in ownership.

If voucher revocations are supported, devices MAY dynamically obtain the voucher revocation artifact (or equivalents) from an Internet based resource. If the access to the Internet based resource is sufficiently reliable, then there may not be a need for the voucher revocation artifact to be supplied by any other means (e.g., Section 6). However, since the access may not be sufficiently reliable, support for this artifact is defined herein.

The voucher revocation artifact is used by a device to verify the ownership voucher (Section 4.3) that the device SHOULD have also received, as described in Section 5. In particular, the device verifies that the voucher revocation explicitly states either that the given voucher is valid or that it is not invalid.

In order to validate a voucher revocation artifact, a device MUST verify that it was signed by a private key associated with a trusted certificate known to the device in its factory default state, as described in Section 8.1, and the device MUST verify that the voucher revocation hasn't expired, and the device SHOULD verify that the revocation is sufficiently fresh, per local policy.

The voucher revocation artifact, including its encoding, is formally defined in [draft-kwatsen-netconf-voucher].

4.6. Certificate Revocation

The certificate revocation artifact is a list of CRLS used to verify the revocation status of owner certificates. Certificate revocations are signed by the certificate authority (or delegate) that issued the owner certificate.

Certificate revocations are generally needed when it is critical for devices to know that assurances implied at the time the certificate was signed are still valid at the time the certificate is being processed.

The need for devices to insist on verifying certificate revocation status is a decision for each manufacturer. If certificate revocation status verification is not asserted, then the owner certificates are essentially forever, which may be acceptable for various kinds of devices. If revocations are supported, then it becomes possible to support various scenarios such as handling a key compromise or expiration.

If certificate revocations are supported, devices MAY dynamically obtain the certificate revocation artifact from an Internet based resource (using a CRL distribution point or an OCSP responder). If the access to the Internet based resource is sufficiently reliable, then there may not be a need for the certificate revocation artifact to be supplied by any other means (e.g., Section 6). However, since the access may not be sufficiently reliable, support for this artifact is defined herein, so that the voucher revocation artifact can be distributed by any source of bootstrapping data.

The certificate revocation artifact is used by a device to verify the owner certificate (Section 4.4) that the device SHOULD have also received, as described in Section 5. In particular, the device verifies that the certificate revocation explicitly states either that the given certificate is valid or that it is not invalid.

In order to validate the CRLs contained with the certificate revocation artifact, a device MUST verify that the CRL was signed by a private key associated certificate's issuer (or delegate), and the device MUST verify that the CRL hasn't expired, and the device SHOULD verify that the revocation is sufficiently fresh, per local policy.

The certificate revocation artifact is formally an unsigned PKCS #7 SignedData structure as specified by RFC 2315 [RFC2315], Section 9.1, containing just CRLs (no content, signatures, or certificates), encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

The certificate revocation artifact contains, in order, the CRL for the owner certificate itself and the CRLs for all intermediate certificates leading up to but not including a trust anchor certificate.

5. Artifact Groupings

Section 4 lists all the possible bootstrapping artifacts, but only certain groupings of these artifacts make sense to return in the various bootstrapping situations described in this document. The remainder of this section identifies these groupings to further clarify how the artifacts are used.

5.1. Unsigned Information

The first grouping of artifacts is for unsigned information. That is, when the information type artifact (Section 4.1) has not been signed.

Unsigned information is useful for cases when transport level security can be used to convey trust (e.g., HTTPS), or when the information can be processed in a provisional manner (i.e. unsigned redirect information).

Conveying unsigned information entails communicating just one of the six artifacts listed in Section 4, namely the information type artifact.

List of artifacts included in this grouping:

- information type

5.2. Signed Information (without Revocations)

The second grouping of artifacts is for when the information type artifact (Section 4.1) has been signed, without any revocation information.

Signed information is needed when the information is obtained from an untrusted source of bootstrapping data (Section 6) and yet it is desired that the device be able to trust the information (i.e. no provisional processing).

Revocation information may not need to be provided because, for instance, the device only uses revocation information obtained dynamically from Internet based resources. Another possible reason may be because the device does not have a reliable clock, and therefore the manufacturer decides to never revoke information (e.g., ownership assignments are forever).

Conveying signed information without revocation information entails communicating four of the six artifacts listed in Section 4.

List of artifacts included in this grouping:

- information type
- signature
- ownership voucher
- owner certificate

5.3. Signed Information (with Revocations)

The third grouping of artifacts is for when the information type artifact (Section 4.1) has been signed and also includes revocation information.

Signed information, as described above, is needed when the information is obtained from an untrusted source of bootstrapping data (Section 6) and yet it is desired that the device be able to trust the information (i.e. no provisional processing).

Revocation information may need to be provided because, for instance, the device insists on being able to verify revocations and the device is deployed on a private network and therefore unable to obtain the revocation information from Internet based resources.

Conveying signed information with revocation information entails communicating all six of the artifacts listed in Section 4.

List of artifacts included in this grouping:

- information type
- signature
- ownership voucher
- owner certificate
- voucher revocations
- certificate revocations

6. Sources of Bootstrapping Data

This section defines some sources for zero touch bootstrapping data that a device can access. The list of sources defined here is not meant to be exhaustive. It is left to future documents to define additional sources for obtaining zero touch bootstrapping data.

For each source defined in this section, details are given for how each of the six artifacts listed in Section 4 is provided.

6.1. Removable Storage

A directly attached removable storage device (e.g., a USB flash drive) MAY be used as a source of zero touch bootstrapping data.

To use a removable storage device as a source of bootstrapping data, a device need only detect if the removable storage device is plugged in and mount its filesystem.

Use of a removable storage device is compelling, as it doesn't require any external infrastructure to work. It is also compelling

that the raw boot image file can be located on the removable storage device, enabling a removable storage device to be a fully self-standing bootstrapping solution.

A removable storage device is an untrusted source of bootstrapping data. This means that the information stored on the removable storage device either **MUST** be signed, or it **MUST** be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a removable storage device presents itself as a file-system, the bootstrapping artifacts need to be presented as files. The six artifacts defined in Section 4 are mapped to files below.

Artifact to File Mapping:

Information Type: Mapped to a file containing a standard YANG encoding for the YANG modeled data described in Section 4.1. A filenames convention **SHOULD** be used to indicate data encoding (e.g., boot-info.[xml|json]).

Signature: Mapped to a file containing the binary artifact described in Section 4.2.

Ownership Voucher: Mapped to a file containing the binary artifact described in Section 4.3.

Owner Certificate: Mapped to a file containing the binary artifact described in Section 4.4.

Voucher Revocation: Mapped to a file containing the binary artifact described in Section 4.5.

Certificate Revocation: Mapped to a file containing binary artifact described in Section 4.6.

The format of the removable storage device's filesystem and the naming of the files are outside the scope of this document. However, in order to facilitate interoperability, it is **RECOMMENDED** devices support open and/or standards based filesystems. It is also **RECOMMENDED** that devices assume a filenames convention that enables more than one instance of bootstrapping data to exist on a removable storage device. The filenames convention **SHOULD** be unique to the manufacturer, in order to enable bootstrapping data from multiple manufacturers to exist on a removable storage device.

6.2. DNS Server

A DNS server MAY be used as a source of zero touch bootstrapping data.

Using a DNS server may be a compelling option for deployments having existing DNS infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

To use a DNS server as a source of bootstrapping data, a device MAY perform a multicast DNS [RFC6762] query searching for the service "_zerotouch._tcp.local.". Alternatively the device MAY perform DNS-SD [RFC6763] via normal DNS operation, using the domain returned to it from the DHCP server; for example, searching for the service "_zerotouch._tcp.example.com".

Unsigned DNS records (not using DNSSEC as described in [RFC6698]) are an untrusted source of bootstrapping data. This means that the information stored in the DNS records either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a DNS server presents resource records (Section 3.2.1 of [RFC1035]), the bootstrapping artifacts need to be presented as resource records. The six artifacts defined in Section 4 are mapped to resource records below.

Artifact to Resource Record Mapping:

Information Type: Mapped to a TXT record called "info-type" containing a standard YANG encoding for the YANG modeled data described in Section 4.1. Note: no additional field is provided to specify the encoding.

Signature: Mapped to a TXT record called "sig" containing the base64-encoding of the binary artifact described in Section 4.2.

Ownership Voucher: Mapped to a TXT record called "voucher" containing the base64-encoding of the binary artifact described in Section 4.3.

Owner Certificate: Mapped to a TXT record called "cert" containing the base64-encoding of the binary artifact described in Section 4.4.

Voucher Revocation: Mapped to a TXT record called "vouch-rev" containing the base64-encoding of the binary artifact described in Section 4.5.

Certificate Revocation: Mapped to a TXT record called "cert-rev" that containing the base64-encoding of the binary artifact described in Section 4.6.

TXT records have an upper size limit of 65535 bytes (Section 3.2.1 in RFC1035), since 'RDLENGTH' is a 16-bit field. Please see Section 3.1.3 in RFC4408 for how a TXT record can achieve this size. Due to this size limitation, some information type artifacts may not fit. In particular, the bootstrap information artifact could hit this upper bound, depending on the size of the included configuration and scripts.

When bootstrap information is provided, it is notable that the URL for the boot-image the device can download would have to point to another server (e.g., http://, ftp://, etc.), as DNS servers do not themselves distribute files.

6.3. DHCP Server

A DHCP server MAY be used as a source of zero touch bootstrapping data.

To use a DHCP server as a source of bootstrapping data, a device need only send a DHCP lease request to a DHCP server. However, the device SHOULD pass the Vendor Class Identifier (option 60) field in its DHCP lease request, so the DHCP server can return bootstrap information shared by devices from the same vendor. However, if it is desired to return device-specific bootstrap information, then the device SHOULD also send the Client Identifier (option 61) field in its DHCP lease request, so the DHCP server can select the specific bootstrap information that has been staged for that one device.

Using a DHCP server may be a compelling option for deployments having existing DHCP infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

A DHCP server is an untrusted source of bootstrapping data. This means that the information returned by the DHCP server either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a DHCP server presents data as DHCP options, the bootstrapping artifacts need to be presented as

DHCP options, specifically the ones specified in Section 11.1. The six artifacts defined in Section 4 are mapped to the DHCP options specified in Section 11.1 below.

Artifact to DHCP Option Field Mapping:

Information Type: Mapped to the DHCP option field "information-type" containing the YANG modeled data described in Section 4.1. The additional field "encoding" is provided to specify the encoding used, taking the values "xml" or "json".

Signature: Mapped to the DHCP option field "signature" containing the binary artifact described in Section 4.2.

Ownership Voucher: Mapped to the DHCP option field "ownership-voucher" containing the binary artifact described in Section 4.3.

Owner Certificate: Mapped to the DHCP option field "owner-certificate" containing the binary artifact described in Section 4.4.

Voucher Revocation: Mapped to the DHCP option field "voucher-revocations" containing the binary artifact described in Section 4.5.

Certificate Revocation: Mapped to the DHCP option field "certificate-revocations" containing the binary artifact described in Section 4.6.

When bootstrap information is provided, it is notable that the URL for the boot-image the device can download would have to point to another server (e.g., http://, ftp://, etc.), as DHCP servers do not themselves distribute files.

6.4. Bootstrap Server

A bootstrap server MAY be used as a source of zero touch bootstrapping data. A bootstrap server is defined as a RESTCONF ([draft-ietf-netconf-restconf]) server implementing the YANG module provided in Section 9.

Unlike any other source of bootstrap data described in this document, a bootstrap server is not only a source of data, but it can also receive data from devices using the YANG-defined "notification" action statement defined in the YANG module (Section 9.2). The data sent from devices both enables visibility into the bootstrapping process (e.g., warnings and errors) as well as provides potentially

useful completion status information (e.g., the device's SSH host-keys).

To use a bootstrap server as a source of bootstrapping data, a device MUST use the RESTCONF protocol to access the YANG container node /device/, passing its own serial number in the URL as the key to the 'device' list.

Using a bootstrap server as a source of bootstrapping data is a compelling option as it uses transport-level security in lieu of signed data, which may be easier to deploy in some situations. Additionally, the bootstrap server is able to receive notifications from devices, which may be critical to some deployments (e.g., the passing of the device's SSH host keys).

A bootstrap server may be trusted or an untrusted source of bootstrapping data, depending on how the device learned about the bootstrap server's trust anchor from a trusted source. When a bootstrap server is trusted, the information returned from it MAY be signed. However, when the server is untrusted, in order for its information to be of any use to the device, the information MUST either be signed or be information that can be processed provisionally (e.g., unsigned redirect information).

When a device is able to trust a bootstrap server, it MUST send its IDevID certificate in the form of a TLS client certificate, and it MUST send notifications to the bootstrap server. When a device is not able to trust a bootstrap server, it MUST NOT send its IDevID certificate in the form of a TLS client certificate, and it MUST NOT send any notifications to the bootstrap server.

From an artifact perspective, since a bootstrap server presents data as a YANG-modeled data, the bootstrapping artifacts need to be mapped to nodes in the YANG module. The six artifacts defined in Section 4 are mapped to bootstrap server nodes defined in Section 9.2 below.

Artifact to Bootstrap Server Node Mapping:

Information Type: Mapped to the choice node /device/information-type.

Signature: Mapped to the leaf node /device/signature.

Ownership Voucher: Mapped to the leaf node /device/ownership-voucher.

Owner Certificate: Mapped to the leaf node /device/owner-certificate.

Voucher Revocations: Mapped to the leaf node /device/voucher-revocation.

Certificate Revocations: Mapped to the leaf-list node /device/certificate-revocation.

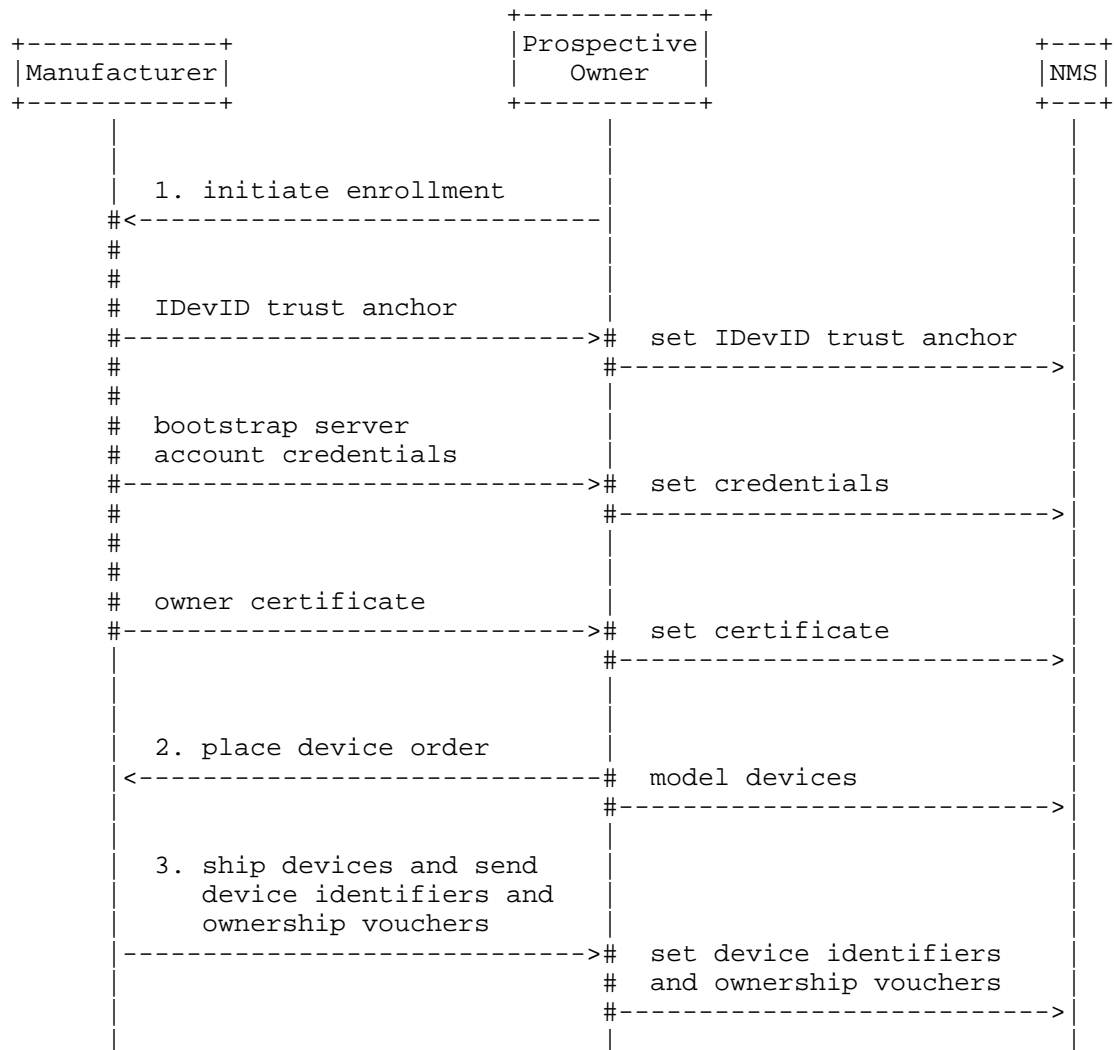
While RESTCONF servers typically support a nested hierarchy of resources, zero touch bootstrap servers only need to support the paths /device and /device/notification. The processing instructions provided in Section 8.3 only uses these two URLs.

7. Workflow Overview

The zero touch solution presented in this document is conceptualized to be composed of the workflows described in this section. Implementations MAY vary in details. Each diagram is followed by a detailed description of the steps presented in the diagram, with further explanation on how implementations may vary.

7.1. Onboarding and Ordering Devices

The following diagram illustrates key interactions that may occur from when a prospective owner enrolls in a manufacturer's zero touch program to when the manufacturer ships devices for an order placed by the prospective owner.



Each numbered item below corresponds to a numbered item in the diagram above.

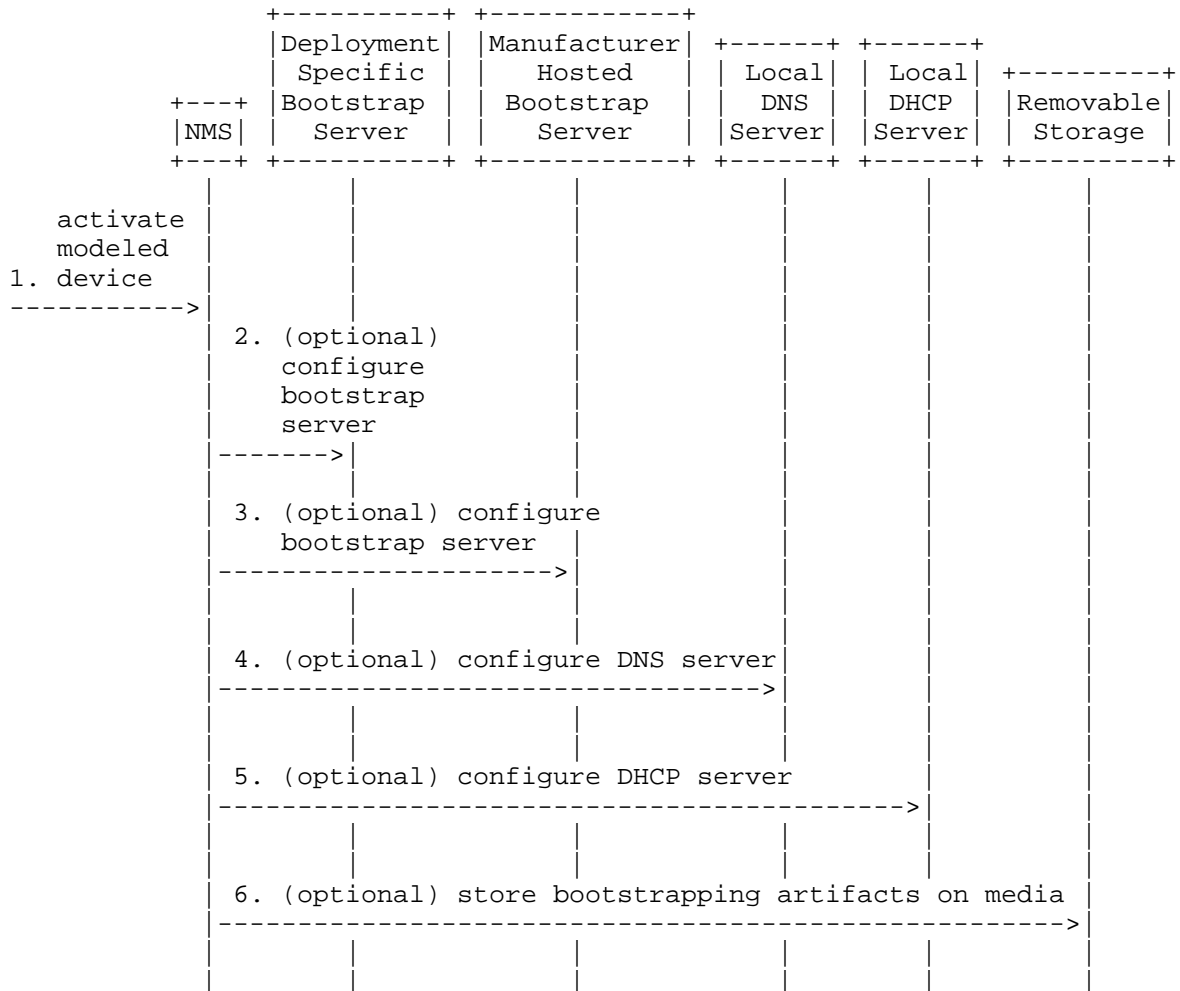
1. A prospective owner of a manufacturer's devices, or an existing owner that wishes to start using zero touch for future device orders, initiates an enrollment process with the manufacturer or delegate. This process includes the following:
 - * Regardless how the prospective owner intends to bootstrap their devices, they will always obtain from the manufacturer or delegate the trust anchor certificate for its device's

IDevID certificates. This certificate will need to be installed on the prospective owner's NMS so that the NMS can subsequently authenticate the device's IDevID certificates.

- * If the manufacturer hosts an Internet based bootstrap server (e.g., a redirect server) such as described in Section 6.4, then credentials necessary to configure the bootstrap server would be provided to the prospective owner. If the bootstrap server is configurable through an API (outside the scope of this document), then the credentials might be installed on the prospective owner's NMS so that the NMS can subsequently configure the manufacturer-hosted bootstrap server directly.
 - * If the manufacturer's devices are able to validate signed data (Section 8.4), then the manufacturer, acting as a certificate authority, may additionally sign an owner certificate for the prospective owner. Alternatively, and not depicted, the owner may obtain an owner certificate from a manufacturer-trusted 3rd-party certificate authority, and report that certificate to the manufacturer. How the owner certificate is used to enable devices to validate signed bootstrapping data is described in Section 8.4. Assuming the prospective owner's NMS is able to prepare and sign the bootstrapping data, the owner certificate would be installed on the NMS at this time.
2. Some time later, the prospective owner places an order with the manufacturer (or delegate), perhaps with a special flag checked for zero touch handling. At this time, or perhaps before placing the order, the owner may model the devices in their NMS, creating virtual objects for the devices with no real-world device associations. For instance the model can be used to simulate the device's location in the network and the configuration it should have when fully operational.
 3. When the manufacturer or delegate fulfills the order, shipping the devices to their intended locations, they may notify the owner of the devices's unique identifiers (e.g., serial numbers) and shipping destinations, which the owner may use to stage the network for when the devices power on. Additionally, the manufacturer may send one or more ownership vouchers, cryptographically assigning ownership of those devices to the rightful owner. The owner may set this information on their NMS, perhaps binding specific modeled devices to the unique identifiers and ownership vouchers.

7.2. Owner Stages the Network for Bootstrap

The following diagram illustrates how an owner might stage the network for bootstrapping devices.



Each numbered item below corresponds to a numbered item in the diagram above.

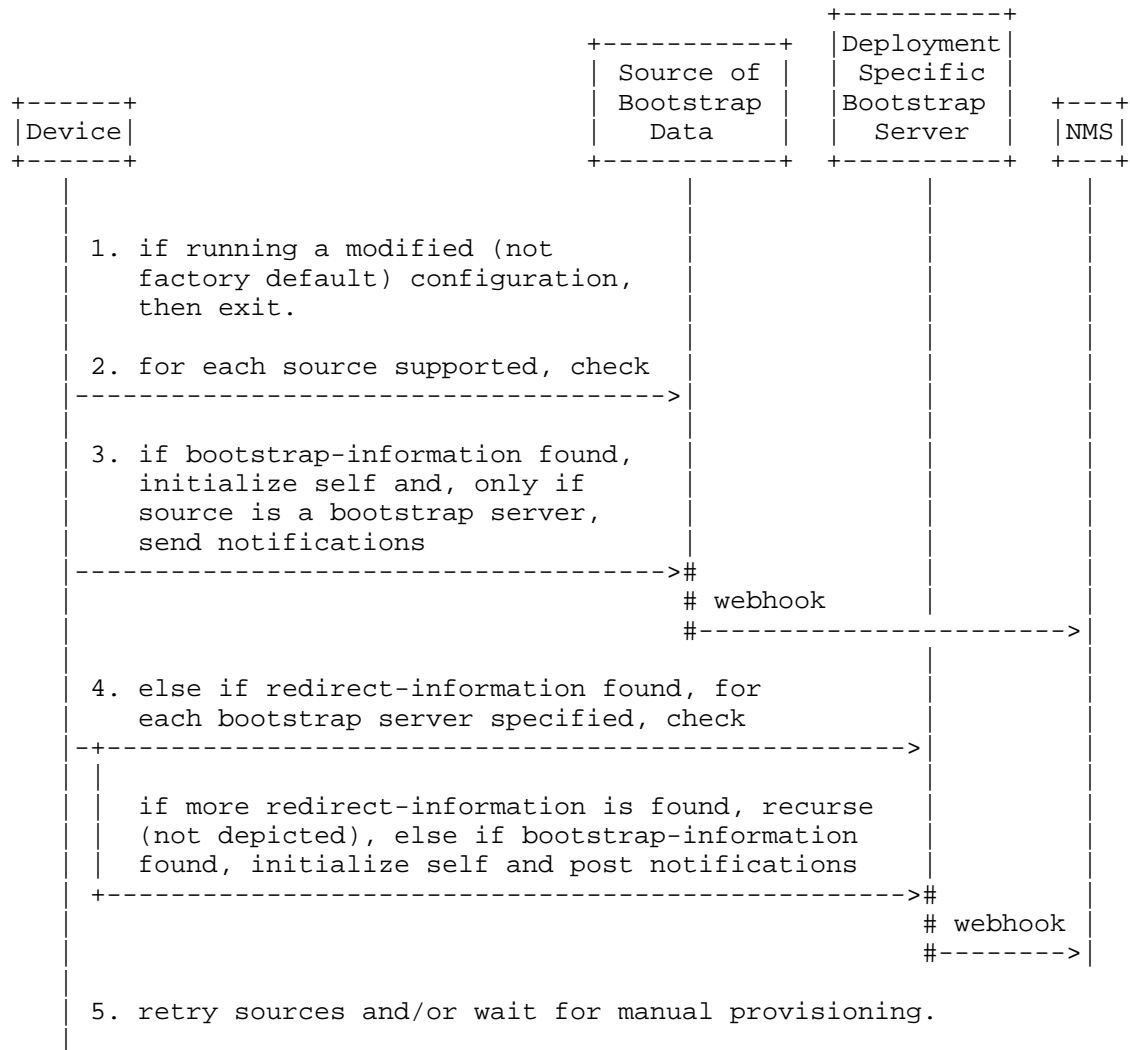
1. Having previously modeled the devices, including setting their fully operational configurations and associating both device identifiers (e.g., serial numbers) and ownership vouchers, the owner "activates" one or more modeled devices. That is, the owner tells the NMS to perform the steps necessary to prepare for

when the real-world devices power up and initiate the bootstrapping process. Note that, in some deployments, this step might be combined with the last step from the previous workflow. Here it is depicted that an NMS performs the steps, but they may be performed manually or through some other mechanism.

2. If it is desired to use a deployment specific bootstrap server, it **MUST** be configured to provide the bootstrapping information for the specific devices. Configuring the bootstrap server **MAY** occur via a programmatic API not defined by this document. Illustrated here as an external component, the bootstrap server **MAY** be implemented as an internal component of the NMS itself.
3. If it is desired to use a manufacturer (or delegate) hosted bootstrap server, it **MUST** be configured to provide the bootstrapping information for the specific devices. The configuration **MUST** be either redirect or bootstrap information. That is, either the manufacturer hosted bootstrap server will redirect the device to another bootstrap server, or provide the device with its bootstrapping information itself. The types of bootstrapping information the manufacturer hosted bootstrap server supports **MAY** vary by implementation; some implementations may only support redirect information, or only support bootstrap information, or support both redirect and bootstrap information. Configuring the bootstrap server **MAY** occur via a programmatic API not defined by this document.
4. If it is desired to use a DNS server to supply bootstrapping information, a DNS server needs to be configured. If multicast DNS-SD is desired, then the server **MUST** reside on the local network, otherwise the DNS server **MAY** reside on a remote network. Please see Section 6.2 for more information about how to configure DNS servers. Configuring the DNS server **MAY** occur via a programmatic API not defined by this document.
5. If it is desired to use a DHCP server to supply bootstrapping data, a DHCP server needs to be configured. The DHCP server may be accessed directly or via a DHCP relay. Please see Section 6.3 for more information about how to configure DHCP servers. Configuring the DHCP server **MAY** occur via a programmatic API not defined by this document.
6. If it is desired to use a removable storage device (e.g., USB flash drive) to supply bootstrapping information, the information would need to be placed onto it. Please see Section 6.1 for more information about how to configure a removable storage device.

7.3. Device Powers On

The following diagram illustrates the sequence of activities that occur when a device powers on.



The interactions in the above diagram are described below.

1. Upon power being applied, the device's bootstrapping logic first checks to see if it is running in its factory default state. If it is in a modified state, then the bootstrapping logic exits and none of the following interactions occur.

2. For each source of bootstrapping data the device supports, preferably in order of closeness to the device (e.g., removable storage before Internet based servers), the device checks to see if there is any bootstrapping data for it there.
3. If bootstrap-information is found, the device initializes itself accordingly (e.g., installing a boot-image and committing an initial configuration). If the source is a bootstrap server, and the bootstrap server can be trusted (i.e., TLS-level authentication), the device also sends progress notifications to the bootstrap server.
 - * The contents of the initial configuration SHOULD configure an administrator account on the device (e.g., username, ssh-rsa key, etc.) and SHOULD configure the device either to listen for NETCONF or RESTCONF connections or to initiate call home connections ([draft-ietf-netconf-call-home]).
 - * If the bootstrap server supports forwarding device notifications to external systems (e.g., via a webhook), the "bootstrap-complete" notification (Section 9.2) informs the external system to know when it can, for instance, initiate a connection to the device (assuming it knows the device's address and the device was configured to listen for connections). To support this further, the bootstrap-complete notification also relays the device's SSH host keys and/or TLS certificates, with which the external system can use to authenticate subsequent connections to the device.

If the device is ever able to complete the bootstrapping process successfully (i.e., no longer running its factory default configuration), it exits the bootstrapping logic without considering any additional sources of bootstrapping data.

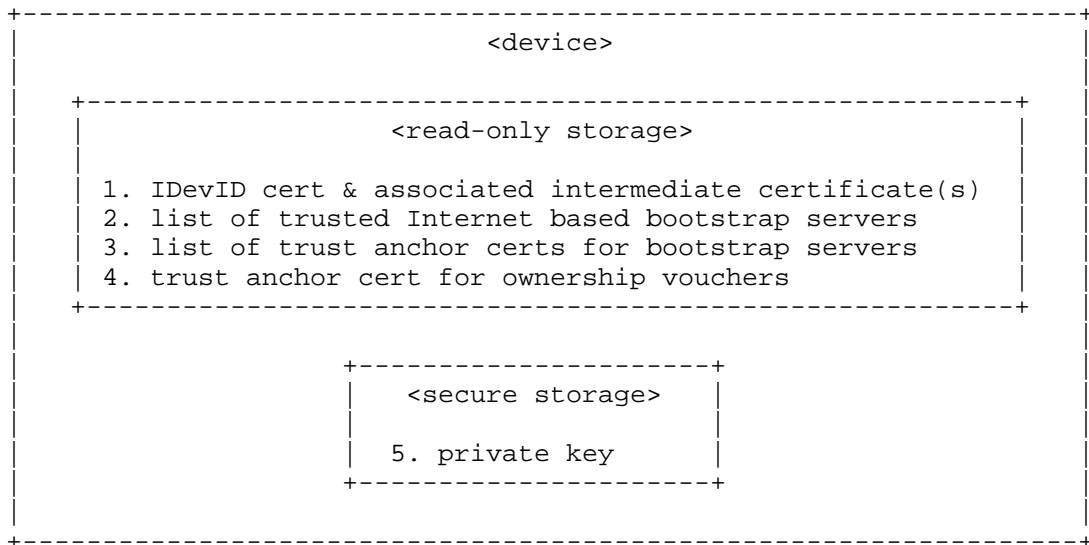
4. Otherwise, if redirect-information is found, the device iterates through the list of specified bootstrap servers, checking to see if there is any bootstrapping data for it on them. If the bootstrap server returns more redirect-information, then the device processes it recursively. Otherwise, if the bootstrap server returns bootstrap-information, the device processes it following the description provided in (3) above.
5. After having tried all supported sources of bootstrapping data, the device MAY retry again all the sources and/or provide manageability interfaces for manual configuration (e.g., CLI, HTTP, NETCONF, etc.). If manual configuration is allowed, and such configuration is provided, the device MUST immediately cease

trying to obtain bootstrapping data, as it would then no longer be in running its factory default configuration.

8. Device Details

Devices supporting the bootstrapping strategy described in this document MUST have the preconfigured factory default state and bootstrapping logic described in the following sections.

8.1. Factory Default State



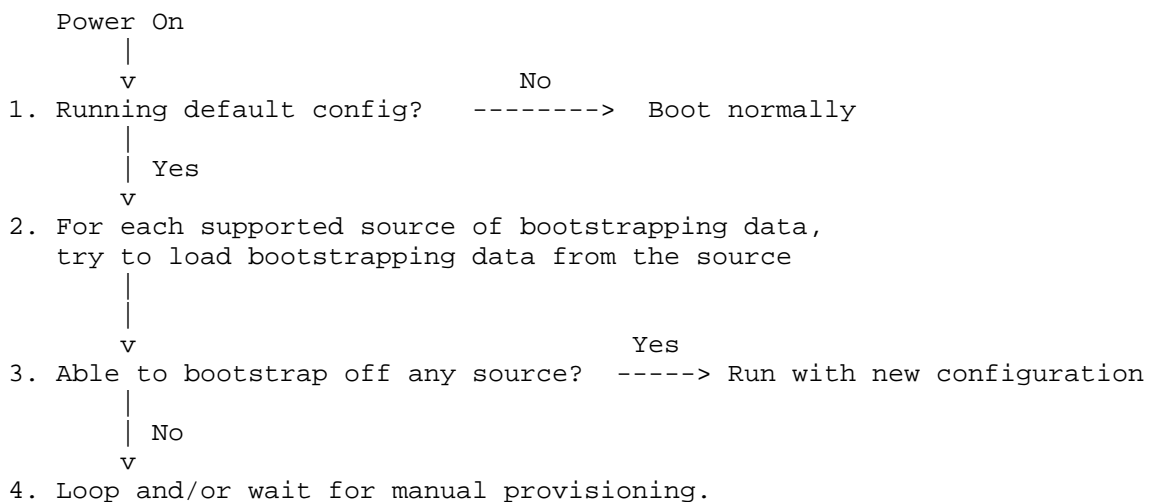
Each numbered item below corresponds to a numbered item in the diagram above.

1. Devices MUST be manufactured with an initial device identifier (IDevID), as defined in [Std-802.1AR-2009]. The IDevID is an X.509 certificate, encoding the device's unique device identifier (e.g., serial number). The device MUST also possess any intermediate certificates between the IDevID certificate and the manufacturer's IDevID trust anchor certificate, which is provided to prospective owners separately (e.g., Section 7.1).
2. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 6.4) MUST be manufactured with a configured list of trusted bootstrap servers. Consistent with redirect information (Section 3.1, each bootstrap server MAY be identified by its hostname or IP address, and an optional port.

3. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 6.4) MUST also be manufactured with a list of trust anchor certificates that can be used for X.509 certificate path validation ([RFC6125], Section 6) on the bootstrap server's TLS server certificate.
4. Devices that support loading owner signed data (see Section 1.2) MUST also be manufactured with the trust anchor certificate for the ownership vouchers.
5. Device MUST be manufactured with a private key that corresponds to the public key encoded in the device's IDevID certificate. This private key SHOULD be securely stored, ideally by a cryptographic processor (e.g., a TPM).

8.2. Boot Sequence

A device claiming to support the bootstrapping strategy defined in this document MUST support the boot sequence described in this section.



Each numbered item below corresponds to a numbered item in the diagram above.

1. When the device powers on, it first checks to see if it is running the factory default configuration. If it is running a modified configuration, then it boots normally.

2. The device iterates over its list of sources for bootstrapping data (Section 6). Details for how to process a source of bootstrapping data are provided in Section 8.3.
3. If the device is able to bootstrap itself off any of the sources of bootstrapping data, it runs with the new bootstrapped configuration.
4. Otherwise the device MAY loop back through the list of bootstrapping sources again and/or wait for manual provisioning.

8.3. Processing a Source of Bootstrapping Data

This section describes a recursive algorithm that a device claiming to support the bootstrapping strategy defined in this document MUST use to authenticate bootstrapping data. A device enters this algorithm for each new source of bootstrapping data. The first time the device enters this algorithm, it MUST initialize a conceptual trust state variable, herein referred to as "trust-state", to FALSE. The ultimate goal of this algorithm is for the device to process bootstrap information (Section 3.2) while the trust-state variable is TRUE.

If the data source is a bootstrap server, and the device is able to authenticate the server using X.509 certificate path validation ([RFC6125], Section 6) to one of the device's preconfigured trust anchors, or to a trust anchor that it learned from a previous step, then the device MUST set trust-state to TRUE.

If trust-state is TRUE, when connecting to the bootstrap server, the device MUST use its IDevID certificate for client certificate based authentication and MUST POST progress notifications using the bootstrap server's "notification" action. Otherwise, if trust-state is FALSE, when connecting to the bootstrap server, the device MUST NOT use its IDevID certificate for a client certificate based authentication and MUST NOT POST progress notifications using the bootstrap server's "notification" action.

When accessing a bootstrap server, the device MUST only access its top-level resource, to obtain all the data staged for it in one GET request, so that it can determine if the data is signed or not, and thus act accordingly. If trust-state is TRUE, then the device MAY also access the bootstrap servers 'notification' resource for the device.

For any source of bootstrapping data (e.g., Section 6), if the data is signed and the device is able to validate the signed data using the algorithm described in Section 8.4, then the device MUST set

trust-state to TRUE, else the device MUST set trust-state to FALSE. Note, this is worded to cover the special case when signed data is returned even from a trusted bootstrap server.

If the data is bootstrap information (not redirect information), and trust-state is FALSE, the device MUST exit the recursive algorithm, returning to the state machine described in Section 8.2. Otherwise, the device MUST attempt to process the bootstrap information as described in Section 8.6. In either case, success or failure, the device MUST exit the recursive algorithm, returning to the state machine described in Section 8.2, the only difference being in how it responds to the "Able to bootstrap off any source?" conditional described in that state machine.

If the data is redirect information, the device MUST process the redirect information as described in Section 8.5. This is the recursion step, it will cause the device to reenter this algorithm, but this time the data source will most definitely be a bootstrap server, as that is all redirect information is able to do.

8.4. Validating Signed Data

Whenever a device is presented signed data from an untrusted source, it MUST validate the signed data as described in this section. If the signed data is provided by a trusted source, a redundant trust case, the device MAY skip verifying the signature.

Whenever there is signed data, the device MUST also be provided an ownership voucher and an owner certificate. Depending on circumstances, the device MAY also be provided certificate and voucher revocations. How all the needed artifacts are provided for each source of bootstrapping data is defined in Section 6.

The device MUST first authenticate the ownership voucher by validating the signature on it to one of its preconfigured trust anchors (see Section 8.1) and verify that the voucher contains the device's unique identifier (e.g., serial number). If the device insists on verifying revocation status, it MUST also verify that the voucher isn't expired or has been revoked. If the authentication of the voucher is successful, the device extracts the rightful owner's identity from the voucher for use in the next step.

Next the device MUST authenticate the owner certificate by performing X.509 certificate path validation on it, and by verifying that the certificate is both identified by the voucher and also has in its chain of trust the certificate identified by the voucher. If the device insists on verifying revocation status, it MUST also verify that none of the certificates in the chain of certificates have been

revoked or expired. If the authentication of the certificate is successful, the device extracts the owner's public key from the certificate for use in the next step.

Finally the device MUST verify the signature over 'information type' artifact was generated by the private key matching the public key extracted from the owner certificate in the previous step.

When the device receives the signed data from a bootstrap server, the device MUST use text-level operations to extract the 'information-type' node from the parent 'device' node in the response in order to verify the signature. It is not important if the extracted text is a valid YANG encoding in order to verify the signature.

If any of these steps fail, then the device MUST mark the data as invalid and not perform any of the subsequent steps.

8.5. Processing Redirect Information

In order to process redirect information (Section 3.1), the device MUST follow the steps presented in this section.

Processing redirect information is straightforward. The device sequentially steps through the list of provided bootstrap servers until it can find one it can bootstrap off of.

If a hostname is provided, and the hostname's DNS resolution is to more than one IP address, the device MUST attempt to connect to all of the DNS resolved addresses at least once, before moving on to the next bootstrap server. If the device is able to obtain bootstrapping data from any of the DNS resolved addresses, it MUST immediately process that data, without attempting to connect to any of the other DNS resolved addresses.

If the redirect information is trusted (e.g., trust-state is TRUE), and the bootstrap server entry contains a trust anchor certificate, then the device MUST authenticate the bootstrap server using X.509 certificate path validation ([RFC6125], Section 6) to the specified trust anchor. If the device is unable to authenticate the bootstrap server to the specified trust anchor, the device MUST NOT attempt a provisional connection to the bootstrap server (i.e., by blindly accepting its server certificate).

If the redirect information is untrusted (e.g., trust-state is FALSE), the device MUST discard any trust anchors provided by the redirect information and establish a provisional connection to the bootstrap server (i.e., by blindly accepting its TLS server certificate).

8.6. Processing Bootstrap Information

In order to process bootstrap information (Section 3.2), the device MUST follow the steps presented in this section.

When processing bootstrap information, the device MUST first process the boot image information, then execute the pre-configuration script (if any), then commit the initial configuration, and then execute the script (if any), in that order. If the device encounters an error at any step, it MUST NOT proceed to the next step.

First the device MUST determine if the image it is running satisfies the specified boot image criteria (e.g., name or fingerprint match). If it does not, the device MUST download, verify, and install the specified boot image, and then reboot. To verify the boot image, the device MUST check that the boot image file matches the fingerprint (e.g., sha256) supplied by the bootstrapping information. Upon rebooting, the device MUST still be in its factory default state, causing the bootstrapping process to run again, which will eventually come to this very point, but this time the device's running image will satisfy the specified criteria, and thus the device will move to processing the next step.

Next, for devices that support executing scripts, if a pre-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if had any warnings or errors. In the case of errors, the device MUST reset itself in such a way that force the reinstallation of its boot image, thereby wiping out any bad state the script might have left behind.

Next the device commits the provided initial configuration. Assuming no errors, the device moves to processing the next step.

Again, for devices that support executing scripts, if a post-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if it had any warnings or errors. In the case of errors, the device MUST reset itself in such a way that force the reinstallation of its boot image, thereby wiping out any bad state the script might have left behind.

At this point, the device has completely processed the bootstrapping data and is ready to be managed. If the device obtained the bootstrap information from a trusted bootstrap server, the device MUST send the 'bootstrap-complete' notification now.

At this point the device is configured and no longer running its factory default configuration. Notably, if the bootstrap information

configured the device it initiate a call home connection, the device would proceed to do so now.

9. RESTCONF API for Bootstrap Servers

This section defines a YANG ([RFC6020]) module that is used to define the RESTCONF ([draft-ietf-netconf-restconf]) API used by the bootstrap server defined in Section 6.4. Examples illustrating this API in use are provided in Appendix A.

9.1. Tree Diagram

The following tree diagram provides an overview for the bootstrap server RESTCONF API. The syntax used for this tree diagram is described in Section 1.4.

```

module: ietf-zerotouch-bootstrap-server
  +--ro device* [unique-id]
    +--ro unique-id          string
    +--ro (information-type)
      +--:(redirect-information)
        +--ro redirect-information
          +--ro bootstrap-server* [address]
            +--ro address      inet:host
            +--ro port?        inet:port-number
            +--ro trust-anchor? binary
          +--:(bootstrap-information)
            +--ro bootstrap-information
              +--ro boot-image
                +--ro name      string
                +--ro (hash-algorithm)
                  +--:(sha256)
                    +--ro sha256? string
                +--ro uri*      inet:uri
              +--ro configuration-handling? enumeration
              +--ro pre-configuration-script? script
              +--ro configuration?
              +--ro post-configuration-script? script
            +--ro signature?    binary
          +--ro ownership-voucher? binary
          +--ro owner-certificate? binary
          +--ro voucher-revocation? binary
          +--ro certificate-revocation? binary
        +---x notification
          +---w input
            +---w notification-type    enumeration
            +---w message?             string
            +---w ssh-host-keys
              +---w ssh-host-key*
                +---w format            enumeration
                +---w key-data          string
            +---w trust-anchors
              +---w trust-anchor*
                +---w protocol*         enumeration
                +---w certificate       binary

```

In the above diagram, notice that all of the protocol accessible nodes are read-only, to assert that devices can only pull data from the bootstrap server.

Also notice that the module defines an action statement, which devices use to provide progress notifications to the bootstrap server.

9.2. YANG Module

The bootstrap server's device-facing API is normatively defined by the YANG module defined in this section.

Note: the module defined herein uses data types defined in [RFC2315], [RFC5280], [RFC6234], [RFC6991], and [draft-kwatsen-netconf-voucher].

<CODE BEGINS> file "ietf-zerotouch-bootstrap-server@2016-10-31.yang"

```
module ietf-zerotouch-bootstrap-server {
  yang-version "1.1";

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server";
  prefix "ztbs";

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

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

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>
    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>";

  description
    "This module defines an interface for bootstrap servers, as defined
    by RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
    Management.

    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 XXXX; see the RFC
    itself for full legal notices.";
```

```
revision "2016-10-31" {
  description
    "Initial version";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
    Management";
}

list device {
  key unique-id;
  config false;

  description
    "A device's record entry. This is the only RESTCONF resource
    that a device will GET, as described in Section 8.2 in RFC XXXX.
    Getting just this top-level node provides a device with all the
    data it needs in a single request.";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF or
    RESTCONF based Management";

  leaf unique-id {
    type string;
    description
      "A unique identifier for the device (e.g., serial number).
      Each device accesses its bootstrapping record by its unique
      identifier.";
  }

  choice information-type {
    mandatory true;
    description
      "This choice statement ensures the response only contains
      redirect-information or bootstrap-information. Note that
      this is the only mandatory true node, as the other nodes
      are not needed when the device trusts the bootstrap server,
      in which case the data does not need to be signed.";

    container redirect-information {
      description
        "This is redirect information, as described in Section 3.1
        in RFC XXXX. Its purpose is to redirect a device to another
        bootstrap server.";
      reference
        "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF
        based Management";
    }
  }
}
```

```
list bootstrap-server {
  key address;
  description
    "A bootstrap server entry.";

  leaf address {
    type inet:host;
    mandatory true;
    description
      "The IP address or hostname of the bootstrap server the
       device should redirect to.";
  }
  leaf port {
    type inet:port-number;
    default 443;
    description
      "The port number the bootstrap server listens on.";
  }
  leaf trust-anchor { //should there be two fields like voucher?
    type binary;
    description
      "An X.509 v3 certificate structure as specified by RFC
       5280, Section 4, encoded using ASN.1 distinguished
       encoding rules (DER), as specified in ITU-T X.690. A
       certificate that a device can use as a trust anchor
       to authenticate the bootstrap server it is being
       redirected to.";
    reference
      "RFC 5280:
       Internet X.509 Public Key Infrastructure Certificate
       and Certificate Revocation List (CRL) Profile.
       ITU-T X.690:
       Information technology - ASN.1 encoding rules:
       Specification of Basic Encoding Rules (BER),
       Canonical Encoding Rules (CER) and Distinguished
       Encoding Rules (DER).";
  }
}

container bootstrap-information {

  description
    "This is bootstrap information, as described in Section 3.2 in
     RFC XXXX. Its purpose is to provide the device everything it
     needs to bootstrap itself.";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF"
```

```
    based Management";

container boot-image {
  description
    "Specifies criteria for the boot image the device MUST
    be running.";

  leaf name { // maybe this should be a regex?
    type string;
    mandatory true;
    description
      "The name of a software image that the device MUST
      be running in order to process the remaining nodes.";
  }
  choice hash-algorithm {
    mandatory true;
    description
      "Identifies the hash algorithm used.";
    leaf sha256 {
      type string;
      description
        "The hex-encoded SHA-256 hash over the boot
        image file. This is used by the device to
        verify a downloaded boot image file.";
      reference
        "RFC 6234: US Secure Hash Algorithms.";
    }
  }
  leaf-list uri {
    type inet:uri;
    min-elements 1;
    description
      "An ordered list of URIs to where the boot-image file MAY
      be obtained. Deployments MUST know in which URI schemes
      (http, ftp, etc.) a device supports. If a secure scheme
      (e.g., https) is provided, a device MAY establish a
      provisional connection to the server, by blindly
      accepting the server's credentials (e.g., its TLS
      certificate)";
  }
}

leaf configuration-handling {
  type enumeration {
    enum merge {
      description
        "Merge configuration into existing running configuration.";
    }
  }
}
```



```
        enum replace {
            description
                "Replace existing running configuration with the passed
                configuration.";
        }
    }
    description
        "This enumeration indicates how the server should process
        the provided configuration.  When not specified, the device
        MAY determine how to process the configuration using other
        means (e.g., vendor-specific metadata).";
}

leaf pre-configuration-script {
    type script;
    description
        "A script that, when present, is executed before the
        configuration has been processed.";
}

anydata configuration {
    must "../configuration-handling";
    description
        "Any configuration data model known to the device.  It may
        contain manufacturer-specific and/or standards-based data
        models.";
}

leaf post-configuration-script {
    type script;
    description
        "A script that, when present, is executed after the
        configuration has been processed.";
}
}

leaf signature {
    type binary;
    must "../redirect-information or ../bootstrap-information" {
        description
            "An information type must be present whenever an
            signature is present.";
    }
    description
        "A PKCS#7 SignedData structure, as specified by Section 9.1
        of RFC 2315, containing just the signature (no content,
        certificates, or CRLs), encoded using ASN.1 distinguished
```

encoding rules (DER), as specified in ITU-T X.690.

This signature is generated by the device's owner using the private key associated with the owner certificate over the information-type node, exactly as it's presented to the device. The device MUST use text-level operations to extract the information-type node from the larger 'device' response in order to verify it. It is not important if the extracted text is itself a valid encoding (e.g., XML or JSON).";

reference

"RFC 2315:

PKCS #7: Cryptographic Message Syntax Version 1.5

ITU-T X.690:

Information technology - ASN.1 encoding rules:
Specification of Basic Encoding Rules (BER),
Canonical Encoding Rules (CER) and Distinguished
Encoding Rules (DER).";

}

leaf ownership-voucher {

type binary;

must "../signature" {

description

"A signature must be present whenever an ownership voucher is presented.";

}

must "../owner-certificate" {

description

"An owner certificate must be present whenever an ownership voucher is presented.";

}

description

"A 'voucher' structure, per draft-kwatsen-netconf-voucher. The voucher needs to reference the device's unique identifier and also specify the owner certificate's identity and a CA certificate in the owner certificate's chain of trust.";

reference

"draft-kwatsen-netconf-voucher:

Voucher and Voucher Revocation Profiles for Bootstrapping
Protocols";

}

leaf owner-certificate {

type binary;

must "../signature" {

description

"A signature must be present whenever an owner certificate

```
        is presented.";
    }
    must "../ownership-voucher" {
        description
            "An ownership voucher must be present whenever an owner
            certificate is presented.";
    }
    description
        "An unsigned PKCS #7 SignedData structure, as specified
        by Section 9.1 in RFC 2315, containing just certificates
        (no content, signatures, or CRLs), encoded using ASN.1
        distinguished encoding rules (DER), as specified in
        ITU-T X.690.

        This structure contains, in order, the owner certificate
        itself and all intermediate certificates leading up to a
        trust anchor certificate. The owner certificate MAY
        optionally include the trust anchor certificate.";
    reference
        "RFC 2315:
        PKCS #7: Cryptographic Message Syntax Version 1.5.
        ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
}

leaf voucher-revocation {
    type binary;
    must "../ownership-voucher" {
        description
            "An ownership voucher must be present whenever a voucher
            revocation is presented.";
    }
    description
        "A 'voucher-revocation' structure, as defined in
        draft-kwatsen-netconf-voucher. The voucher revocation
        definitively states whether a voucher is valid or not.";
    reference
        "draft-kwatsen-netconf-voucher:
        Voucher and Voucher Revocation Profiles for Bootstrapping
        Protocols";
}

leaf certificate-revocation {
    type binary;
    must "../owner-certificate" {
```

```
    description
      "An owner certificate must be present whenever an voucher
        revocation is presented.";
  }
  description
    "An unsigned PKCS #7 SignedData structure, as specified by
      Section 9.1 in RFC 2315, containing just CRLs (no content,
      signatures, or certificates), encoded using ASN.1
      distinguished encoding rules (DER), as specified in
      ITU-T X.690.

      This structure contains, in order, the CRL for the owner
      certificate itself and the CRLs for all intermediate
      certificates leading up to but not including a trust
      anchor certificate.";
  reference
    "RFC 5280:
      Internet X.509 Public Key Infrastructure Certificate
      and Certificate Revocation List (CRL) Profile.
      ITU-T X.690:
      Information technology - ASN.1 encoding rules:
      Specification of Basic Encoding Rules (BER),
      Canonical Encoding Rules (CER) and Distinguished
      Encoding Rules (DER).";
}

action notification {
  input {
    leaf notification-type {
      type enumeration {
        enum bootstrap-initiated {
          description
            "Indicates that the device has just accessed the
              bootstrap server. The 'message' field below MAY
              contain any additional information that the
              manufacturer thinks might be useful.";
        }
        enum validation-error {
          description
            "Indicates that the device had an issue validating
              the response from the bootstrap server. The
              'message' field below SHOULD indicate the specific
              error. This message also indicates that the device
              has abandoned trying to bootstrap off this bootstrap
              server.";
        }
        enum signature-validation-error {
          description
```

```
"Indicates that the device had an issue validating the
bootstrapping data. For instance, this could be due
to the device expecting signed data, but only found
unsigned data, or because the ownership voucher didn't
include the device's unique identifier, or because the
signature didn't match. The 'message' field below
SHOULD indicate the specific error. This message also
indicates that the device has abandoned trying to
bootstrap off this bootstrap server.";
}
enum image-mismatch {
  description
    "Indicates that the device has determined that its
    running image does not match the specified criteria.
    The 'message' field below SHOULD indicate both what
    image the device is currently running.";
}
enum image-download-error {
  description
    "Indicates that the device had an issue downloading
    the image, which could be for reasons such as a file
    server being unreachable to the downloaded file
    being the incorrect file (signature mismatch). The
    'message' field about SHOULD indicate the specific
    error. This message also indicates that the device
    has abandoned trying to bootstrap off this bootstrap
    server.";
}
enum pre-script-warning {
  description
    "Indicates that the device obtained a greater than
    zero exit status code from the script when it was
    executed. The 'message' field below SHOULD indicate
    both the resulting exit status code, as well as
    capture any stdout/stderr messages the script may
    have produced.";
}
enum pre-script-error {
  description
    "Indicates that the device obtained a less than zero
    exit status code from the script when it was executed.
    The 'message' field below SHOULD indicate both the
    resulting exit status code, as well as capture any
    stdout/stderr messages the script may have produced.
    This message also indicates that the device has
    abandoned trying to bootstrap off this bootstrap
    server.";
}
```

```
enum config-warning {
  description
    "Indicates that the device obtained warning messages
    when it committed the initial configuration. The
    'message' field below SHOULD indicate the warning
    messages that were generated.";
}
enum config-error {
  description
    "Indicates that the device obtained error messages
    when it committed the initial configuration. The
    'message' field below SHOULD indicate the error
    messages that were generated. This message also
    indicates that the device has abandoned trying to
    bootstrap off this bootstrap server.";
}
enum post-script-warning {
  description
    "Indicates that the device obtained a greater than
    zero exit status code from the script when it was
    executed. The 'message' field below SHOULD indicate
    both the resulting exit status code, as well as
    capture any stdout/stderr messages the script may
    have produced.";
}
enum post-script-error {
  description
    "Indicates that the device obtained a less than zero
    exit status code from the script when it was executed.
    The 'message' field below SHOULD indicate both the
    resulting exit status code, as well as capture any
    stdout/stderr messages the script may have produced.
    This message also indicates that the device has
    abandoned trying to bootstrap off this bootstrap
    server.";
}
enum bootstrap-complete {
  description
    "Indicates that the device successfully processed the
    all the bootstrapping data and that it is ready to
    be managed. The 'message' field below MAY contain
    any additional information that the manufacturer
    thinks might be useful. After sending this message,
    the device is not expected to access the bootstrap
    server again.";
}
enum informational {
  description
```

```
        "Indicates any additional information not captured by
        any of the other notification-type. The 'message'
        field below SHOULD contain any additional information
        that the manufacturer thinks might be useful.";
    }
}
mandatory true;
description
    "The type of notification provided.";
}
leaf message {
    type string;
    description
        "An optional human-readable value.";
}
container ssh-host-keys {
    description
        "A list of SSH host keys an NMS may use to authenticate
        a NETCONF connection to the device with.";
    list ssh-host-key {
        when "../notification-type = 'bootstrap-complete'" {
            description
                "SSH host keys are only sent when the notification
                type is 'bootstrap-complete'.";
        }
        description
            "An SSH host-key";
        leaf format {
            type enumeration {
                enum ssh-dss { description "ssh-dss"; }
                enum ssh-rsa { description "ssh-rsa"; }
            }
            mandatory true;
            description
                "The format of the SSH host key.";
        }
        leaf key-data {
            type string;
            mandatory true;
            description
                "The key data for the SSH host key";
        }
    }
}
container trust-anchors {
    description
        "A list of trust anchor certificates an NMS may use to
        authenticate a NETCONF or RESTCONF connection to the
```

```
        device with.";
    list trust-anchor {
        when "../notification-type = 'bootstrap-complete'" {
            description
                "Trust anchors are only sent when the notification
                 type is 'bootstrap-complete'.";
        }
        description
            "A list of trust anchor certificates an NMS may use to
             authenticate a NETCONF or RESTCONF connection to the
             device with.";
        leaf-list protocol {
            type enumeration {
                enum netconf-ssh      { description "netconf-ssh"; }
                enum netconf-tls     { description "netconf-tls"; }
                enum restconf-tls    { description "restconf-tls"; }
                enum netconf-ch-ssh   { description "netconf-ch-ssh"; }
                enum netconf-ch-tls   { description "netconf-ch-tls"; }
                enum restconf-ch-tls { description "restconf-ch-tls"; }
            }
            min-elements 1;
            description
                "The protocols that this trust anchor secures.";
        }
        leaf certificate {
            type binary;
            mandatory true;
            description
                "An X.509 v3 certificate structure, as specified by
                 Section 4 in RFC5280, encoded using ASN.1 distinguished
                 encoding rules (DER), as specified in ITU-T X.690.";
            reference
                "RFC 5280:
                 Internet X.509 Public Key Infrastructure Certificate
                 and Certificate Revocation List (CRL) Profile.
                 ITU-T X.690:
                 Information technology - ASN.1 encoding rules:
                 Specification of Basic Encoding Rules (BER),
                 Canonical Encoding Rules (CER) and Distinguished
                 Encoding Rules (DER).";
        }
    }
}
} // end action
} // end device
```



```
typedef script {  
    type binary;  
    description  
        "A device specific script that enables the execution of commands  
        to perform actions not possible thru configuration alone.  
  
        No attempt is made to standardize the contents, running context,  
        or programming language of the script. The contents of the  
        script are considered specific to the vendor, product line,  
        and/or model of the device.  
  
        If a script is erroneously provided to a device that does not  
        support the execution of scripts, the device SHOULD send a  
        'script-warning' notification message, but otherwise continue  
        processing the bootstrapping data as if the script had not  
        been present.  
  
        The script returns exit status code '0' on success and non-zero  
        on error, with accompanying stderr/stdout for logging purposes.  
        In the case of an error, the exit statuscode will specify what  
        the device should do.  
  
        If the exit status code is greater than zero, then the device  
        should assume that the script had a soft error, which the  
        script believes does not affect manageability. If the device  
        obtained the bootstrap information from a bootstrap server,  
        it SHOULD send a 'script-warning' notification message.  
  
        If the exit status code is less than zero, the device should  
        assume the script had a hard error, which the script believes  
        will affect manageability. In this case, the device SHOULD  
        send a 'script-error' notification message followed by a  
        reset that will force a new boot-image install (wiping out  
        anything the script may have done) and restart the entire  
        bootstrapping process again.";  
}  
}  
  
<CODE ENDS>
```

10. Security Considerations

10.1. Immutable storage for trust anchors

Devices **MUST** ensure that all their trust anchor certificates, including those for connecting to bootstrap servers and verifying ownership vouchers, are protected from external modification.

It may be necessary to update these certificates over time (e.g., the manufacturer wants to delegate trust to a new CA). It is therefore expected that devices **MAY** update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

10.2. Clock Sensitivity

The solution in this document relies on TLS certificates, owner certificates, and ownership vouchers, all of which require an accurate clock in order to be processed correctly (e.g., to test validity dates and revocation status). Implementations **MUST** ensure devices have an accurate clock when shipped from manufacturing facilities, and take steps to prevent clock tampering.

If it is not possible to ensure clock accuracy, it is **RECOMMENDED** that implementations disable the aspects of the solution having clock sensitivity. In particular, such implementations should assume that TLS certificates, owner certificates, and ownership vouchers are not revokable. In real-world terms, this means that manufacturers **SHOULD** only issue a single ownership voucher for the lifetime of some devices.

It is important to note that implementations **SHOULD NOT** rely on NTP for time, as it is not a secure protocol.

10.3. Blindly authenticating a bootstrap server

This document allows a device to blindly authenticate a bootstrap server's TLS certificate. It does so to allow for cases where the redirect information may be obtained in an unsecured manner, which is desirable to support in some cases.

To compensate for this, this document requires that devices, when connected to an untrusted bootstrap server, do not send their IDevID certificate for client authentication, and they do not POST any progress notifications, and they assert that data downloaded from the server is signed.

10.4. Entropy loss over time

Section 7.2.7.2 of the IEEE Std 802.1AR-2009 standard says that IDevID certificate should never expire (i.e. having the notAfter value 99991231235959Z). Given the long-lived nature of these certificates, it is paramount to use a strong key length (e.g., 512-bit ECC).

10.5. Serial Numbers

This draft suggests using the device's serial number as the unique identifier in its IDevID certificate. This is because serial numbers are ubiquitous and prominently contained in invoices and on labels affixed to devices and their packaging. That said, serial numbers many times encode revealing information, such as the device's model number, manufacture date, and/or sequence number. Knowledge of this information may provide an adversary with details needed to launch an attack.

10.6. Sequencing Sources of Bootstrapping Data

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed for security reasons, as the solution for each source is considered equally secure. However, from a privacy perspective, it is RECOMMENDED that devices access local sources before accessing remote sources.

11. IANA Considerations

11.1. The BOOTP Manufacturer Extensions and DHCP Options Registry

The following registrations are in accordance to RFC 2939 [RFC2939] for "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at <http://www.iana.org/assignments/bootp-dhcp-parameters>.

11.1.1. DHCP v4 Option

Tag: XXX

Name: Zero Touch Information

Returns up to six zero touch bootstrapping artifacts.

Code	Len
XXX	n
encoding	information-type
signature	
owner-certificate	ownership-voucher
certificate-revocations	
voucher-revocations	

Reference: RFC XXXX

11.1.2. DHCP v6 Option

Tag: YYY

Name: Zero Touch Information

Returns up to six zero touch bootstrapping artifacts.

Code	Len
XXX	n
encoding	information-type
signature	
owner-certificate	ownership-voucher
certificate-revocations	
voucher-revocations	

Reference: RFC XXXX

11.2. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

11.3. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format defined in [RFC6020], the the following registration is requested:

name: ietf-zerotouch-bootstrap-server
namespace: urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server
prefix: ztbs
reference: RFC XXXX

12. Other Considerations

Both this document and [draft-ietf-anima-bootstrapping-keyinfra] define bootstrapping mechanisms. The authors have collaborated on both solutions and believe that each solution has merit and, in fact, can work together. That is, it is possible for a device to support both solutions simultaneously.

13. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): David Harrington, Michael Behringer, Dean Bogdanovic, Martin Bjorklund, Joe Clarke, Toerless Eckert, Stephen Farrell, Stephen Hanna, Wes Hardaker, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Phil Shafer, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original I-D's solution during the IETF 87 meeting in Berlin.

14. References

14.1. Normative References

- [draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-10 (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-ietf-netconf-restconf-10>>.
- [draft-kwatsen-netconf-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "Voucher and Voucher Revocation Profiles for Bootstrapping Protocols", draft-kwatsen-netconf-voucher-00 (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-kwatsen-netconf-voucher>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <<http://www.rfc-editor.org/info/rfc2315>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<http://www.rfc-editor.org/info/rfc6125>>.

- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<http://www.rfc-editor.org/info/rfc6234>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <<http://www.rfc-editor.org/info/rfc7468>>.
- [Std-802.1AR-2009]
IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.

14.2. Informative References

- [draft-ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Behringer, M., and S. Bjarnason, "Bootstrapping Key Infrastructures", draft-ietf-anima-bootstrapping-keyinfra-03 (work in progress), 2016, <<https://datatracker.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra>>.
- [draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home (work in progress)", draft-ietf-netconf-restconf-10 (work in progress), December 2015, <<https://datatracker.ietf.org/html/draft-ietf-netconf-call-home-17>>.
- [draft-ietf-netconf-server-model]
Watsen, K., "NETCONF Server Model (work in progress)", draft-ietf-netconf-server-model-09 (work in progress), March 2016, <<https://datatracker.ietf.org/html/draft-ietf-netconf-call-home-17>>.

- [RFC2939] Droms, R., "Procedures and IANA Guidelines for Definition of New DHCP Options and Message Types", BCP 43, RFC 2939, DOI 10.17487/RFC2939, September 2000, <<http://www.rfc-editor.org/info/rfc2939>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<http://www.rfc-editor.org/info/rfc6698>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

Appendix A. API Examples

This section presents some examples illustrating device interactions with a bootstrap server to access Redirect and Bootstrap information, both unsigned and signed, as well as to send a progress notification. These examples show the bootstrap information containing configuration from the YANG modules in [RFC7317] and [draft-ietf-netconf-server-model].

A.1. Unsigned Redirect Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives unsigned redirect information. This example is representative of a response a trusted redirect server might return.

REQUEST

[\'\' line wrapping added for formatting only]

```
GET https://example.com/restconf/data/ietf-zero-touch-bootstrap-server:\
device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml
```

RESPONSE

```
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml
```

<!-- \'\' line wrapping added for formatting purposes only -->

<device

```
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zero-touch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <redirect-information>
    <bootstrap-server>
      <address>phs1.example.com</address>
      <port>8443</port>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbWlCdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
        lLQllsdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgprYjk\
        zSFNwSDdwVXBCYnA4dmtNanFtZ jJma3RqZHBxeFppUUtTbndWZTF2Zwot\
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMEdBMVVkRGd\
```

```

VEJiz0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF\
NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC\
Z05WSFI4RVlqQmdNRjZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
MkF6a3hqUDlVQWtHR0dvS1UleUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
25PZnpZNEhONApXY0pTaUpZK2xtYWS3RTRORUZXXZS9RdGp4NULXZmdvN2\
RJSUJQFRStS0Cg==
</trust-anchor>
</bootstrap-server>
<bootstrap-server>
  <address>phs2.example.com</address>
  <port>8443</port>
  <trust-anchor>
    WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
    lLQ1lsdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgprYjk\
    zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
    NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMEdBMVVkRGd\
    VEJiz0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
    V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF\
    NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC\
    Z05WSFI4RVlqQmdNRjZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
    WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
    QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
    MkF6a3hqUDlVQWtHR0dvS1UleUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
    25PZnpZNEhONApXY0pTaUpZK2xtYWS3RTRORUZXXZS9RdGp4NULXZmdvN2\
    RJSUJQFRStS0Cg==
  </trust-anchor>
</bootstrap-server>
</redirect-information>
</device>

```

A.2. Signed Redirect Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives signed redirect information. This example is representative of a response that redirect server might return if concerned the device might not be able to authenticate its TLS certificate.

REQUEST

['\ ' line wrapping added for formatting only]

```

GET https://example.com/restconf/data/ietf-zero-touch-bootstrap-server:\
device=123456 HTTP/1.1
HOST: example.com

```

Accept: application/yang.data+xml

RESPONSE

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\ ' line wrapping added for formatting purposes only -->

```
<device
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zero-touch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <redirect-information>
    <bootstrap-server>
      <address>phs1.example.com</address>
      <port>8443</port>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbWlCdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
        lLQl1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
        zSFNwSDdwVXBCYnA4dmtNanFtZ jJma3RqZHBxeFppUUtTbndWZTF2Zwot\
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMedBMVVkRGd\
        VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNwd4cFJBZ1ZOYUU0cERZd05ER\
        V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF\
        NQmdOVkhSTUJBZ jhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC\
        Z05WSFI4RVlqQmdNRjZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
        WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
        QmdOVkhJBWVRBbFZUTVJBd0RnWURWUWFLRXdkbAp1R0Z0Y0d4bE1RNHdEQ\
        MkF6a3hqUDlVQWtHR0dvSlUleUclSVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
        25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXXZS9RdGp4NUlXZmdvN2\
        RJSUJQFRStS0Cg==
      </trust-anchor>
    </bootstrap-server>
    <bootstrap-server>
      <address>phs2.example.com</address>
      <port>8443</port>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbWlCdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
        lLQl1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
        zSFNwSDdwVXBCYnA4dmtNanFtZ jJma3RqZHBxeFppUUtTbndWZTF2Zwot\
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMedBMVVkRGd\
        VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNwd4cFJBZ1ZOYUU0cERZd05ER\
        V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF\
        NQmdOVkhSTUJBZ jhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSGdEQnBC\
        Z05WSFI4RVlqQmdNRjZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
```

```
WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
QmdOVkJBWVRBbFZUTVJBd0RnWURWUWFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
MkF6a3hqUDlVQWtHR0dvSlUleUclSVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXXS9RdGp4NULXZmdvN2\
RJSUJQFRStS0Cg==
</trust-anchor>
</bootstrap-server>
</redirect-information>
<signature>
RDEuRiZNRNLeJpgN9YwkXLAZX2rASwy041EMmZ6KakWUd3ZmXucfoLpdRemfuPii\
QGplbmmlwZXIuY29tMB4XDTE0MDIyNzE0MTMlMloXDTE1MDIyNzE0MTMlMlowMDET\
MBEGA1UEChQKVFBNX1ZlbnRvcjEjZmBcGA1UEAxQQSnVuaXB1cl9YWfhYWF9DQTCC\
NT0ufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
</signature>
<ownership-voucher>
ChQQSnVuaXB1cl9OZXR3b3JrczEdMBsGA1UECjQUQ2VydGlmawNhdGVfSXNzdWfu\
Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTABBgkqhkiG9w0BCQEWdMnh\
MBEGA1UEChQKVFBNX1ZlbnRvcjEjZmBcGA1UEAxQQSnVuaXB1cl9YWfhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
yh/JaftWYf7m3KBzOdG2MIHfBgNVHSMegdcwgdSAFDS1jCNmTN5b+CDUjJLlyDal\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEWJVUzETMBEGA1UECBMKQ2FsaWZvcmt5pYTES\
MBAGA1UEBxMjU3Vubnl2YWxlMRkwFwYDVQQKFBBkdW5pcGVyX05ldHdvcmZm0w\
GwYDVQQFLFRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
X0FuY2hvcjEjZmBcGCSqGSIb3DQEJARYOY2FAanVuaXB1ci5jb22CCQDUbsEdTn5v\
MjAO
</ownership-voucher>
<owner-certificate>
MIIEExTCCA62gAwIBAgIBATANBgkqhkiG9w0BAQsFADCBqjELMAkGA1UEBhMCVVMx\
EzARBgNVBAGTCKNhbg1mb3JuaWEExEjaQBgNVBACTCVN1bm55dmFsZTEZMBcGA1UE\
ChQQSnVuaXB1cl9OZXR3b3JrczEdMBsGA1UECjQUQ2VydGlmawNhdGVfSXNzdWfu\
Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTABBgkqhkiG9w0BCQEWdMnh\
QGplbmmlwZXIuY29tMB4XDTE0MDIyNzE0MTMlMloXDTE1MDIyNzE0MTMlMlowMDET\
MBEGA1UEChQKVFBNX1ZlbnRvcjEjZmBcGA1UEAxQQSnVuaXB1cl9YWfhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
RDEuRiZNRNLeJpgN9YwkXLAZX2rASwy041EMmZ6KakWUd3ZmXucfoLpdRemfuPii\
aplDgmS3IaYl/s4OOF8yzcyJprm807NyZp+Y9H1U/7Qfp97/KbqwCgkHSz0lnt0X\
KQTpIM/rNrbrkuTmalezFoFS7mrXlXJAsfPlguVcd7sLCyJvegL8pRCCrU9xyKLF\
8u/Qz4s0x0uzcGYh0sd3iWj2l+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
KmORbiKU2GTGZkaCgCjmrWpvrYwLoXv/sf2nPLyK6YjiWsslOJtRO+KzRbs2B18C\
AwEAAAOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVRO0BBYEFHppoyXF\
yh/JaftWYf7m3KBzOdG2MIHfBgNVHSMegdcwgdSAFDS1jCNmTN5b+CDUjJLlyDal\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEWJVUzETMBEGA1UECBMKQ2FsaWZvcmt5pYTES\
MBAGA1UEBxMjU3Vubnl2YWxlMRkwFwYDVQQKFBBkdW5pcGVyX05ldHdvcmZm0w\
GwYDVQQFLFRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
X0FuY2hvcjEjZmBcGCSqGSIb3DQEJARYOY2FAanVuaXB1ci5jb22CCQDUbsEdTn5v\
MjAOBgNVHQ8BAf8EBAMCAgQwQgYDVRO0fBDswOTA3oDWgM4YxaHR0cDovL2Nybc5q\
dW5pcGVyLm5ldD9jYT1kdW5pcGVyX1RydXN0X0FuY2hvc19DQTANBgkqhkiG9w0B\
AQsFAAOCAQEAOud7EBilqQct3t2C4AXta1gGNNwdldLLw0jtk4BMiA91//DzfSkB\
```

```

2AaJtiseLTXsMF6MQwDs1YKkiXKLu7gBZDlJ6NiDwylUnXhi2BDG+MYXQrc6p76K\
z3bsVwZlaJQCdF5sbggclMyrsOu9QirnRzkIv3R8ndJH5K792ztLquulAcMfnK1Y\
NTOUfhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX7WJzEbT/G7MUfo\
Sb+U2PVsQTDWEzUjVnG7vNWYxirnAOZ00XEWWYxHUUJntx6DsbXYuX7DlPkkNr7ir\
96DpOPtX7h8pXXGSDPBXIyvg02aFMphstQ==
</owner-certificate>
<voucher-revocation>
QGplbmmlwZXIuY29tMB4XDTE0MDIyNzE0MTMlMloXDTE1MDIyNzE0MTMlMlowMDET\
MBEGA1UEChQKVFBNX1ZlbnRvcjEzMBCGA1UEAxQQSnVuaXB1cl9YWFhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
RDEuRiZNRNLeJpgN9YwKXLAZX2rASwy041EMmZ6KakWUd3ZmXucfoLpdRemfuPii\
KQTpIM/rNrbrkuTmalezFoFS7mrXlXJAsfPlguVcD7sLCyJvegL8pRCCrU9xyKLF\
8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBByEFHppoyXF\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcml5pYTES\
NTOUfhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
</voucher-revocation>
<certificate-revocation>
Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTABBgkqhkiG9w0BCQEWDMNh\
MBEGA1UEChQKVFBNX1ZlbnRvcjEzMBCGA1UEAxQQSnVuaXB1cl9YWFhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
yh/JaftWYf7m3KBzOdG2MIHfBgNVHSMegdcwgdSAFDS1jCNmTN5b+CDUjJLlyDal\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcml5pYTES\
MBAGA1UEBxMjU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmZMR0w\
GwYDVQQLEFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
X0FuY2hvcjEEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXB1ci5jb22CCQDUbsEdTn5v\
MjAO==
</certificate-revocation>
</device>

```

A.3. Unsigned Bootstrap Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives unsigned bootstrapping information. This example is representative of a response a locally deployed bootstrap server might return.

REQUEST

['\ ' line wrapping added for formatting only]

```

GET https://example.com/restconf/data/ietf-zero-touch-bootstrap-server:\
device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml

```

RESPONSE

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\ ' line wrapping added for formatting purposes only -->

```
<device
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <bootstrap-information>
    <boot-image>
      <name>
        boot-image-v3.2R1.6.img
      </name>
      <md5>
        SomeMD5String
      </md5>
      <shal>
        SomeShalString
      </shal>
      <uri>
        ftp://ftp.example.com/path/to/file
      </uri>
    </boot-image>
  <configuration-handling>merge</configuration-handling>
  <configuration>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication>
        <user>
          <name>admin</name>
          <authorized-key>
            <name>admin's rsa ssh host-key</name>
            <algorithm>ssh-rsa</algorithm>
            <key-data>AAAAB3NzaC1yc2EAAAADAQABAAQDeJMV8zrtsi8CgEsR\
jCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mw\
E1lG9YxLzeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVc\
WAw1lOr9IDGDAuww6G45gLCHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA\
vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jW\
EIuA7LvEJYql4unq4Iog+/+CiumTkmQIWRgIo j4FCzYkO9NvRE6fOSLLf\
gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
          </authorized-key>
        </user>
      </authentication>
    </system>
```

```
<!-- from ietf-netconf-server.yang -->
<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <call-home>
    <application>
      <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>
    </application>
  </call-home>
</netconf-server>
</configuration>
</bootstrap-information>
</device>
```

A.4. Signed Bootstrap Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives signed bootstrap information. This example is representative of a response that bootstrap server might return if concerned the device might not be able to authenticate its TLS certificate.

REQUEST

['\ ' line wrapping added for formatting only]

```
GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml
```

RESPONSE

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\ ' line wrapping added for formatting purposes only -->

```
<device
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <bootstrap-information>
    <boot-image>
      <name>
        boot-image-v3.2R1.6.img
      </name>
      <md5>
        SomeMD5String
      </md5>
      <shal>
        SomeShalString
      </shal>
      <uri>
        /path/to/on/same/bootserver
      </uri>
    </boot-image>
  <configuration>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication>
        <user>
          <name>admin</name>
          <authorized-key>
            <name>admin's rsa ssh host-key</name>
            <algorithm>ssh-rsa</algorithm>
            <key-data>AAAAB3NzaClyc2EAAAADAQABAAQDeJMV8zrtsi8CgEsR\
jCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mw\
E1lG9YxLzeS5p2ngzK6lvikUSqfMukeBohFTTrDZ8bUtrF+HMLlTRnoCVc\
WAw1lOr9IDGAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA\
vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jW\
EIuA7LvEJYql4unq4Iog+/+CiumTkmQIWRgIoJ4FCzYkO9NvRE6fOSLLf\
gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
          </authorized-key>
        </user>
      </authentication>
    </system>
    <!-- from ietf-netconf-server.yang -->
```



```
<netconf-server>  
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">  
    <call-home>  
      <application>  
        <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>  
      </application>  
    </call-home>  
  </netconf-server>  
</configuration>  
</bootstrap-information>  
<signature>  
  RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy04lEMmZ6KAKWUd3ZmXucfoLpdRemfuPii\  
  QGplbm1wZXIuY29tMB4XDTE0MDIyNzE0MTMlMloXDTElMDIyNzE0MTMlMlowMDET\  
  MBEGA1UEChQKVFBNX1ZlbmRvcjEzMBCGA1UEAxQQSnVuaXB1cl9YWfhYWf9DQTCC\  
  NTOufhQsD2t4TYPekzLEiZqSswdB0ApXPcJLQNw8Bw2xN+A9GX=  
</signature>  
<ownership-voucher>  
  ChQQSnVuaXB1cl9OZZXR3b3JrczEdMBsGA1UECxxQUQ2VydgGlmaWNhdGVfSXNzdWFu\  
  Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\  
  MBEGA1UEChQKVFBNX1ZlbmRvcjEzMBCGA1UEAxQQSnVuaXB1cl9YWfhYWf9DQTCC\  
  ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXLmFxIX\  
  yh/JaftWyf7m3KBzOdgd2MIHfBgNVHSMEgdcdwgdsAFDSljCNmTN5b+CDujJLlyDal\  
  WFPaoYGwpIGTMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcmt5PyTES\  
  MBAGA1UEBxBMJU3Vubnl2YWxlMRkwFwYDVQQKFBBkdW5pcGVyX05ldHdvcmtdzMR0w\  
  GwYDVQQFLBRDZXJ0awZpY2F0ZV9Jc3NlYW5jZTEzMBCGA1UEAxQQVFBNX1RydXN0\  
  X0FuY2hvcjEjEdMBsGCsQGSIb3DQeJARYOY2FAanVuaXB1ci5jb22CCQDUbsEdTn5v\  
  MjAO  
</ownership-voucher>  
<owner-certificate>  
  MIIEExtCCA62gAwIBAgIBATANBgkqhkiG9w0BAQsFADCbqjELMAkGA1UEBhMCVVmx\  
  EzARBgNVBAGTCkNhbgGlm3JuaWEExejAQBgNVBACTCVNlbn55dmFsZTEzMBCGA1UE\  
  ChQQSnVuaXB1cl9OZZXR3b3JrczEdMBsGA1UECxxQUQ2VydgGlmaWNhdGVfSXNzdWFu\  
  Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh
```

```
QGplbm1wZXIuY29tMB4XDTE0MDIyNzE0MTM1Ml0XDTE1MDIyNzE0MTM1Ml0wMDEt\
MBEGA1UEChQKVFBX1ZlbnRvcjEzMjcGAlUEAxQQSnVuaXB1cl9YWfhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
RDEuRiZNRNLeJpgN9YwKXLAZX2rASwy041EMmZ6KakWUd3ZmXucfoLpdRemfuPii\
aplDgmS3IaYl/s40OF8yzcYJprm807NyZp+Y9H1U/7Qfp97/KbqwCgkHSz0lnt0X\
KQTpIM/rNrbrkuTmalezFoFS7mrXLXJAsfPlguVcD7sLCyJvegL8pRCCrU9xyKLF\
8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
KmORbiku2GTGZkaCgCjmrWpvrYwLoXv/sf2nPLyK6YjiWssl0JtRO+KzRbs2B18C\
AwEAAAOCAW0wgGFPMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
yh/JaftWYf7m3KBzOdG2MIHfBgNVHSMEdgcwgdSAFDS1jCNmTN5b+CDuJLlyDal\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
MBAGA1UEBxMjU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcm5pYTES\
GwYDVQQLEFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGAlUEAxQQVFBX1RydXN0\
X0FuY2hvcjEEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXB1cl5jb22CCQDUbsEdTn5v\
MjAOBgNVHQ8BAf8EBAMCAgQwQgYDVR0fBDswOTA3ODWgM4YxaHR0cDovL2Nybc5q\
dW5pcGVyLm5ldD9jYT1KdW5pcGVyX1RydXN0X0FuY2hvc19DQTANBgkqhkiG9w0B\
AQSFAAOCAQEAOu7EBilqQcT3t2C4AXtalGNNwdldLLw0jtk4BMiA9l//DZfskB\
2AaJtiseLTXsMF6MQwDs1YKkiXKLu7gBZDlJ6NiDwy1UnXhi2BDG+MYXQrc6p76K\
z3bsVwZlaJQCdF5sbggclMyrsOu9QirnRZkIv3R8ndJH5K792ztLquulAcMfnK1Y\
NTOUfhQsD2t4TYpEkzLEiZqSswdBOaPXPcJLQNW8Bw2xN+A9GX7WJzEbT/G7MUfo\
Sb+U2PVsQTDWEzUjVnG7vNWYxirnAOZ0OXEWYxHUJntx6DsbXYuX7D1PkkNr7ir\
96DpOPTX7h8pxxGSDPBXIyvg02aFmPhstQ==
</owner-certificate>
<voucher-revocation>
QGplbm1wZXIuY29tMB4XDTE0MDIyNzE0MTM1Ml0XDTE1MDIyNzE0MTM1Ml0wMDEt\
MBEGA1UEChQKVFBX1ZlbnRvcjEzMjcGAlUEAxQQSnVuaXB1cl9YWfhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
RDEuRiZNRNLeJpgN9YwKXLAZX2rASwy041EMmZ6KakWUd3ZmXucfoLpdRemfuPii\
KQTpIM/rNrbrkuTmalezFoFS7mrXLXJAsfPlguVcD7sLCyJvegL8pRCCrU9xyKLF\
8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
AwEAAAOCAW0wgGFPMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
NTOUfhQsD2t4TYpEkzLEiZqSswdBOaPXPcJLQNW8Bw2xN+A9GX=
</voucher-revocation>
<certificate-revocation>
Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDMNh\
MBEGA1UEChQKVFBX1ZlbnRvcjEzMjcGAlUEAxQQSnVuaXB1cl9YWfhYWF9DQTCC\
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLMfxI\
yh/JaftWYf7m3KBzOdG2MIHfBgNVHSMEdgcwgdSAFDS1jCNmTN5b+CDuJLlyDal\
WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
MBAGA1UEBxMjU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcm5pYTES\
GwYDVQQLEFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGAlUEAxQQVFBX1RydXN0\
X0FuY2hvcjEEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXB1cl5jb22CCQDUbsEdTn5v\
MjAO==
</certificate-revocation>
</device>
```

A.5. Progress Notifications

The following example illustrates a device using the API to post a notification to a trusted bootstrap server. Illustrated below is the 'bootstrap-complete' message, but the device may send other notifications to the server while bootstrapping (e.g., to provide status updates).

The bootstrap server **MUST NOT** process a notification from a device without first authenticating the device. This is in contrast to when a device is fetching data from the server, a read-only operation, in which case device authentication is not strictly required (e.g., when sending signed information).

In this example, the device sends a notification indicating that it has completed bootstrapping off the data provided by the server. This example illustrates the device sending both its SSH host keys and TLS server certificate to the bootstrap server, which the bootstrap server may, for example, pass to an NMS, as discussed in Section 7.3.

Note that devices that are able to present an IDevID certificate [Std-802.1AR-2009], when establishing SSH or TLS connections, do not need to include its DevID certificate in the bootstrap-complete message. It is unnecessary to send the DevID certificate in this case because the IDevID certificate does not need to be pinned by an NMS in order to be trusted.

REQUEST

[\'\' line wrapping added for formatting only]

```
POST https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
device=123456/notification HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml
```

<!-- \'\' line wrapping added for formatting purposes only -->

```
<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <notification-type>bootstrap-complete</notification-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>
      <format>ssh-rsa</format>
      <key-data>
        AAAAB3NzaC1yc2EAAAADAQABAAQDeJMV8zrtsi8CgEsRCjCzfve2m6\
```

```
zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2MwjE1lG9YxL\
zeS5p2ngzK6lvikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcCWAw1lOr\
9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5vg7SLq\
QFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWqEIuA7\
LvEJYql4unq4Iog+/+CiumTkmQIWRgIoJ4FCzYkO9NvRE6fOSLLf6gakW\
VOZZgQ8929uWjCWlG1qn2mPibp2Go1
</key-data>
</ssh-host-key>
<ssh-host-key>
  <format>ssh-dsa</format>
  <key-data>
    zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2MwjE1lG9YxL\
    zeS5p2ngzK6lvikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcCWAw1lOr\
    9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5vg7SLq\
    AAAAB3NzaC1yc2EAAAADAQABAAQDeJMV8zrtsi8CgEsRCjCzfve2m6\
    QFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWqEIuA7\
    LvEJYql4unq4Iog+/+CiumTkmQIWRgIoJ4FCzYkO9NvRE6fOSLLf6gakW\
    VOZZgQ8929uWjCWlG1qn2mPibp2Go1
  </key-data>
</ssh-host-key>
</ssh-host-keys>
<trust-anchors>
  <trust-anchor>
    <protocol>netconf-ssh</protocol>
    <protocol>netconf-tls</protocol>
    <protocol>restconf-tls</protocol>
    <protocol>netconf-ch-ssh</protocol>
    <protocol>netconf-ch-tls</protocol>
    <protocol>restconf-ch-tls</protocol>
  <certificate>
    WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
    lLQl1sdWpOc jFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJmZgPRYjk\
    zSFNwSDdwVXBCYnA4dmtNanFtZ jJma3RqZHBXeFppUUtTbndWZTF2Zwot\
    NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPck1CMEdBMVVKRGd\
    VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNwd4cFJBZ1ZOYUU0cERZd05ER\
    V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNS1l6UG8zREF\
    NQmdOVkhSTUJBZ jhFCkFqQUFNQTRHQTfVZER3RUIvd1FFQXdJSgdeQnBC\
    Z05WSFI4RVlqQmdNR jZnSXFbZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
    WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
    QmdOVkJBWVRBbFZUTVJBd0RnWURWUWFLRXdkbAp1R0Z0Y0d4bE1RNHdeQ\
    MkF6a3hqUDlVQWtHR0dvS1UleUclSVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
    25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXS9RdGp4NUlXZmdvN2\
    RJSUJQFRStS0Cg==
  </certificate>
</trust-anchor>
</trust-anchors>

</input>
```

RESPONSE

HTTP/1.1 204 No Content
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server

Appendix B. Artifact Examples

This section presents examples for how the 'information type' artifact (Section 4.1) can be encoded into a document that can be distributed outside the bootstrap server's RESETCONF API. The encoding for these artifacts is the same as if an HTTP GET request had been sent to the RESTCONF URL for the specific resource.

These examples show the bootstrap information containing configuration from the YANG modules in [RFC7317] and [draft-ietf-netconf-server-model].

Only examples for information type artifact are provided as the other five artifacts in Section 4 have their own encodings.

B.1. Redirect Information

The following example illustrates how redirect information can be encoded into an artifact.

INSERT _TEXT_FROM_FILE(refs/ex-file-redirect-information.xml)

B.2. Bootstrap Information

The following example illustrates how bootstrap information can be encoded into an artifact.

INSERT _TEXT_FROM_FILE(refs/ex-file-bootstrap-information.xml)

Appendix C. Change Log

C.1. ID to 00

- o Major structural update; the essence is the same. Most every section was rewritten to some degree.
- o Added a Use Cases section
- o Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram

- o Removed support for physical presence or any ability for configlets to not be signed.
- o Defined the Zero Touch Information DHCP option
- o Added an ability for devices to also download images from configuration servers
- o Added an ability for configlets to be encrypted
- o Now configuration servers only have to support HTTP/S - no other schemes possible

C.2. 00 to 01

- o Added boot-image and validate-owner annotations to the "Actors and Roles" diagram.
- o Fixed 2nd paragraph in section 7.1 to reflect current use of anyxml.
- o Added encrypted and signed-encrypted examples
- o Replaced YANG module with XSD schema
- o Added IANA request for the Zero Touch Information DHCP Option
- o Added IANA request for media types for boot-image and configuration

C.3. 01 to 02

- o Replaced the need for a configuration signer with the ability for each NMS to be able to sign its own configurations, using manufacturer signed ownership vouchers and owner certificates.
- o Renamed configuration server to bootstrap server, a more representative name given the information devices download from it.
- o Replaced the concept of a configlet by defining a southbound interface for the bootstrap server using YANG.
- o Removed the IANA request for the boot-image and configuration media types

C.4. 02 to 03

- o Minor update, mostly just to add an Editor's Note to show how this draft might integrate with the draft-pritikin-anima-bootstrapping-keyinfra.

C.5. 03 to 04

- o Major update formally introducing unsigned data and support for Internet-based redirect servers.
- o Added many terms to Terminology section.
- o Added all new "Guiding Principles" section.
- o Added all new "Sources for Bootstrapping Data" section.
- o Rewrote the "Interactions" section and renamed it "Workflow Overview".

C.6. 04 to 05

- o Semi-major update, refactoring the document into more logical parts
- o Created new section for information types
- o Added support for DNS servers
- o Now allows provisional TLS connections
- o Bootstrapping data now supports scripts
- o Device Details section overhauled
- o Security Considerations expanded
- o Filled in enumerations for notification types

C.7. 05 to 06

- o Minor update
- o Added many Normative and Informative references.
- o Added new section Other Considerations.

C.8. 06 to 07

- o Minor update
- o Added an Editorial Note section for RFC Editor.
- o Updated the IANA Considerations section.

C.9. 07 to 08

- o Minor update
- o Updated to reflect review from Michael Richardson.

C.10. 08 to 09

- o Added in missing "Signature" artifact example.
- o Added recommendation for manufacturers to use interoperable formats and file naming conventions for removable storage devices.
- o Added configuration-handling leaf to guide if config should be merged, replaced, or processed like an edit-config/yang-patch document.
- o Added a pre-configuration script, in addition to the post-configuration script from -05 (issue #15).

C.11. 09 to 10

- o Factored ownership voucher and voucher revocation to a separate document: draft-kwatsen-netconf-voucher. (issue #11)
- o Removed <configuration-handling> options 'edit-config' and yang-patch'. (issue #12)
- o Defined how a signature over signed-data returned from a bootstrap server is processed. (issue #13)
- o Added recommendation for removable storage devices to use open/standard file systems when possible. (issue #14)
- o Replaced notifications "script-[warning/error]" with "[pre/post]-script-[warning/error]". (goes with issue #15)
- o switched owner-certificate to be encoded using the pkcs#7 format. (issue #16)

- o Replaced md5/sha1 with sha256 inside a choice statement, for future extensibility. (issue #17)
- o A ton of editorial changes, as I went thru the entire draft with a fine-toothed comb.

C.12. 10 to 11

- o fixed yang validation issues found by IETF YANGPageCompilation. note: these issues were NOT found by pyang --ietf or by the submission-time validator...
- o fixed a typo in the yang module, someone the config false statement was removed.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Mikael Abrahamsson
T-Systems

EMail: "mikael.abrahamsson@t-systems.se

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

K. Watsen
Juniper Networks
M. Richardson
SSW
M. Pritikin
T. Eckert
Cisco Systems
October 31, 2016

Voucher and Voucher Revocation Profiles for Bootstrapping Protocols
draft-kwatsen-netconf-voucher-00

Abstract

This memo defines the two artifacts "voucher" and "voucher-revocation", which are YANG-defined structures that have been signed by a TBD algorithm.

The voucher artifact is generated by the device's manufacture or delegate. The voucher's purpose is to securely assign one or more devices to an owner. The voucher informs each device which entity it should consider to be its owner.

The voucher revocation artifact is used by the manufacturer or delegate (i.e. the issuer of the voucher) to revoke vouchers, if ever necessary. The voucher revocation format defined herein supports both issuer-wide and voucher-specific constructs, enabling usage flexibility.

For both artifacts, this memo only defines the artifact, leaving it to future work to describe specialized protocols for accessing them.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Requirements Language	3
3. Tree Diagram Notation	3
4. Voucher	4
4.1. Tree Diagram	4
4.2. Examples	4
4.3. YANG Module	5
5. Voucher Revocation	9
5.1. Tree Diagram	9
5.2. Examples	10
5.3. YANG Module	11
6. Security Considerations	16
6.1. Clock Sensitivity	16
7. IANA Considerations	16
7.1. The IETF XML Registry	16
7.2. The YANG Module Names Registry	17
8. Acknowledgements	17
9. References	17
9.1. Normative References	17
9.2. Informative References	17
Appendix A. Change Log	19
Authors' Addresses	19

1. Introduction

This document defines a strategy to securely assign devices to an owner, using an artifact signed, directly or indirectly, by the device's manufacturer. This artifact is known as the voucher.

A voucher may be useful in several contexts, but the driving motivation herein is to support secure bootstrapping mechanisms, such as are defined in [draft-ietf-netconf-zero-touch] and [draft-ietf-anima-bootstrapping-keyinfra]. Assigning ownership is important to bootstrapping mechanisms so that the booting device can authenticate the network that's trying to take control of it.

The lifetimes of vouchers may vary. In some bootstrapping protocols the vouchers may be ephemeral, whereas in others the vouchers may be potentially long-lived. In order to support the second category of vouchers, this document also defines a voucher revocation artifact, enabling the manufacturer or delegate to communicate the validity of its vouchers.

For both artifacts, this memo only defines the artifact, leaving it to future work to describe specialized protocols for accessing them.

This document uses YANG [RFC7950] to define the voucher and voucher revocation formats. YANG is a data modeling language with established mappings to XML and JSON, with mappings to other encodings in progress. Which encodings a particular solution uses is outside the scope of this document.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the sections below are to be interpreted as described in RFC 2119 [RFC2119].

3. Tree Diagram Notation

The meaning of the symbols in the above diagram is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" (read-write) represents configuration data and "ro" (read-only) represents state data.
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o Ellipsis ("...") stands for contents of subtrees that are not shown.

4. Voucher

The voucher is generated by the device's manufacture or delegate. The voucher's purpose is to securely assign one or more devices to an owner. The voucher informs each device which entity it should consider to be its owner.

The voucher is signed by the device's manufacturer or delegate.

NOTE: AT THIS TIME, THE SIGNING STRATEGY HAS NOT BEEN SELECTED.

4.1. Tree Diagram

Following is the tree diagram for the YANG module specified in Section 4.3. Details regarding each node in the tree diagram are provided in the YANG module. Please see Section 3 for information on tree diagram notation.

```
module: ietf-voucher
  +--ro voucher
    +--ro assertion          enumeration
    +--ro trusted-ca-certificate?  binary
    +--ro certificate-id
      | +--ro cn-id?    string
      | +--ro dns-id?   string
    +--ro unique-id*          string
    +--ro nonce?              string
    +--ro created-on?         yang:date-and-time
    +--ro expires-on?        yang:date-and-time
    +--ro revocation-location? inet:uri
    +--ro additional-data?
```

4.2. Examples

The following illustrates an ephemeral voucher encoded in JSON:

```
{
  "ietf-voucher:voucher": {
    "assertion": "logged",
    "trusted-ca-certificate": "base64-encoded X.509 DER",
    "owner-id": "Registrar3245",
    "unique-id": "JADA123456789",
    "created-on": "2016-10-07T19:31:42Z",
    "nonce": "987987623489567"
  }
}
```

The following illustrates a long-lived voucher encoded in XML:

```
<voucher
  xmlns="urn:ietf:params:xml:ns:yang:ietf-voucher">
  <assertion>verified</assertion>
  <trusted-ca-certificate>
    base64-encoded X.509 DER
  </trusted-ca-certificate>
  <certificate-id>
    <cn-id>Example Inc.</cn-id>  <!-- maybe this should be a DN? -->
    <dns-id>example.com</dns-id>
  </certificate-id>
  <unique-id>AAA123456789</unique-id>
  <unique-id>BBB123456789</unique-id>
  <unique-id>CCC123456789</unique-id>
  <created-on>2016-10-07T19:31:42Z</created-on>
</voucher>
```

4.3. YANG Module

```
module ietf-voucher {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-voucher";
  prefix "vch";

  import ietf-yang-types { prefix yang; }
  import ietf-inet-types { prefix inet; }

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

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>
    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>
    Author:     Max Pritikin
                <mailto:pritikin@cisco.com>
    Author:     Michael Richardson
                <mailto:mcr+ietf@sandelman.ca>";

  description
    "This module defines the format for a voucher, which is
    produced by a device's manufacturer or delegate to securely
    assign one or more devices to an 'owner', so that the
    devices may establish a secure connection to the owner's
```

```
    network infrastructure.";

revision "2016-10-31" {
    description
        "Initial version";
    reference
        "RFC XXXX: Voucher and Voucher Revocation Profiles
        for Bootstrapping Protocols";
}

// top-level container
container voucher {
    config false;
    description
        "A voucher that can be used to assign one or more devices to
        an owner.";

    leaf assertion {
        type enumeration {
            enum verified {
                description
                    "Indicates that the ownership has been positively
                    verified by the device's manufacturer or delegate
                    (e.g., through sales channel integration).";
            }
            enum logged {
                description
                    "Indicates that this ownership assignment has been
                    logged into a database maintained by the device's
                    manufacturer or delegate (voucher transparency).";
            }
        }
    }
    mandatory true;
    description
        "The assertion is a statement from the manufacturer or
        delegate regarding the nature of this voucher. This
        allows the device to know what assurance the manufacturer
        provides, which supports more detailed policy checks
        such as 'I only want to allow verified devices, not
        just logged devices'.";
}

leaf trusted-ca-certificate {
    type binary;
    description
        "An X.509 v3 certificate structure as specified by RFC 5280,
        Section 4 encoded using the ASN.1 distinguished encoding
        rules (DER), as specified in ITU-T X.690.
```

This certificate is used by a bootstrapping device to trust another public key infrastructure, in order to verify another certificate supplied to the device separately by the bootstrapping protocol, the other certificate must have this certificate somewhere in its chain of certificates.";

reference

"RFC 5280:

Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

ITU-T X.690:

Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).";

}

container certificate-id {

description

"When provided, the device MUST also perform RFC 6125 style validation of another certificate supplied to the device separately by the bootstrapping protocol against all the provided ids.";

leaf cn-id {

type string;

description

"The common name field in the certificate must match this value.";

}

leaf dns-id {

type string;

description

"A subjectAltName entry of type dNSName in the certificate must match this value.";

}

}

leaf-list unique-id {

type string;

min-elements 1;

description

"A regular expression identifying one more more device unique identifiers (e.g., serial numbers). For instance, the expression could match just a single serial number, or it might match a range of serial numbers. Devices use this value to determine if the voucher applies to them.";


```
    // Ed. both the zerotouch and brwsky solutions are devid
    // oriented, and so renaming this field to 'serial-number'
    // wouldn't be crazy. But devid/serial-number (typically)
    // assumes physical chassis, is it worth using this
    // term which might extend to e.g. virtual appliances?
}

leaf nonce {
    type string; // unit64?
    description
        "what can be said about this that's ANIMA-neutral?";
}

leaf created-on {
    type yang:date-and-time;
    description
        "The date this voucher was created";
}

leaf expires-on {
    type yang:date-and-time;
    description
        "An optional date value for when this voucher expires.";
}

leaf revocation-location {
    type inet:uri;
    description
        "A URI indicating where revocation information may be obtained.";
}

anydata additional-data {
    description
        "Additional data signed by the manufacturer. The manufacturer
        might put additional data into its vouchers, for human
        consumption or device consumption.";

    // Ed. is the additional data normative? - if so, should we
    // remove this free-form field, and assume it will be formally
    // extended later? Note: the zerotouch draft doesn't need this
    // field...
}
}
```

5. Voucher Revocation

The vouchers revocation artifact is used to verify the revocation status of vouchers. Voucher revocations are signed by the manufacturer or delegate (i.e. the issuer of the voucher). Vouchers revocation statements MAY be verified by devices during the bootstrapping process, or at any time before or after by any entity (e.g., registrar or equivalent) as needed. Registrars or equivalent SHOULD verify voucher revocation statements and make policy decisions in case devices are not doing so themselves.

Revocations are generally needed when it is critical for devices to know that assurances implied at the time the voucher was signed are still valid at the time the voucher is being processed.

As mentioned in Section 1, the lifetimes of vouchers may vary. In some bootstrapping protocols the vouchers may be ephemeral, whereas in others the vouchers may be potentially long-lived. For bootstrapping protocols that support ephemeral vouchers, there is no need to support revocations. For bootstrapping protocols that support long-lived vouchers, the need to support revoking vouchers is a decision for each manufacturer.

If revocations are not supported then voucher assignments are essentially forever, which may be acceptable for various kinds of devices. If revocations are supported, then it becomes possible to support various scenarios such as handling a key compromise or change in ownership.

The voucher revocation format defined herein supports both issuer-wide (similar to a CRL) or voucher-specific (similar to an OSCP response) constructs, enabling usage flexibility.

NOTE: AT THIS TIME, THE SIGNING STRATEGY HAS NOT BEEN SELECTED.

5.1. Tree Diagram

Following is the tree diagram for the YANG module specified in Section 5.3. Details regarding each node in the tree diagram are provided in the YANG module. Please see Section 3 for information on tree diagram notation.

```

module: ietf-voucher-revocation
  +--ro voucher-revocation
    +--ro revocation-type      enumeration
    +--ro created-on           yang:date-and-time
    +--ro expires-on?         yang:date-and-time
    +--ro (voucher-revocation-type)?
      +--:(issuer-wide)
        +--ro issuer-wide
          +--ro (list-type)?
            +--:(whitelist)
              +--ro whitelist
                +--ro voucher-identifier*  string
            +--:(blacklist)
              +--ro blacklist
                +--ro voucher-identifier*  string
          +--:(voucher-specific)
            +--ro voucher-specific
              +--ro voucher-identifier      string
              +--ro voucher-status          enumeration
              +--ro revocation-information
                +--ro revoked-on            yang:date-and-time
                +--ro revocation-reason     enumeration
            +--ro additional-data?

```

5.2. Examples

The following illustrates an issuer-wide voucher revocation in XML:

```

<voucher-revocation
  xmlns="urn:ietf:params:xml:ns:yang:ietf-voucher-revocation">
  <revocation-type>issuer-wide</revocation-type>
  <created-on>2016-10-31T23:59:59Z</created-on>
  <expires-on>2016-12-31T23:59:59Z</expires-on>
  <issuer-wide>
    <blacklist>
      <voucher-identifier>some fingerprint</voucher-identifier>
      <voucher-identifier>some fingerprint</voucher-identifier>
      <voucher-identifier>some fingerprint</voucher-identifier>
    </blacklist>
  </issuer-wide>
</voucher>

```

The following illustrates a voucher-specific revocation in JSON:

```
{
  "ietf-voucher-revocation:voucher-revocation": {
    "revocation-type": "voucher-specific",
    "created-on": "2016-10-31T23:59:59Z"
    "expires-on": "2016-12-31T23:59:59Z"
    "voucher-specific": [
      "voucher-identifier": "some fingerprint",
      "voucher-status": "revoked",
      "revocation-information": [
        "revoked-on": "2016-11-31T23:59:59Z",
        "revocation-reason": "key-compromise"
      ]
    ]
  }
}
```

5.3. YANG Module

```
module ietf-voucher-revocation {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-voucher-revocation";
  prefix "vr";

  import ietf-yang-types { prefix yang; }

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

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>
    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>
    Author:     Max Pritikin
                <mailto:pritikin@cisco.com>
    Author:     Michael Richardson
                <mailto:mcr+ietf@sandelman.ca>";

  description
    "This module defines the format for a voucher revocation,
    which is produced by a manufacturer or delegate to indicate
    the revocation status of vouchers.";

  revision "2016-10-31" {
    description
      "Initial version";
  }
}
```

```
reference
  "RFC XXXX: Voucher and Voucher Revocation Profiles
  for Bootstrapping Protocols";
}

// top-level container
container voucher-revocation {
  config false;
  description
    "A voucher revocation that can provide revocation status
    information for one or more devices.";

  leaf revocation-type {
    type enumeration {
      enum issuer-wide {
        description
          "Indicates that this revocation spans all
          the vouchers the issuer has issued to date";
      }
      enum voucher-specific {
        description
          "Indicated that this revocation only regards
          a single voucher.";
      }
    }
    mandatory true;
    description
      "The revocation-type indicates if the revocation
      is issuer-wide or voucher-specific. Both variations
      exist to enable implementations to choose between the
      number of revocation artifacts generated versus
      individual artifact size.";
  }

  leaf created-on {
    type yang:date-and-time;
    mandatory true;
    description
      "The date this voucher was created";
  }

  leaf expires-on {
    type yang:date-and-time;
    description
      "An optional date value for when this voucher expires.";
  }

  choice voucher-revocation-type {
```

```
description
  "Identifies the revocation type as being either issuer-wide
  or voucher-specific.";

container issuer-wide {
  description
    "This revocation provides issuer-wide revocation status
    (similar to a CRL).";

  choice list-type {
    description
      "Identifies if this issuer-wide revocation is provided
      in the form of a whitelist or a blacklist";

    container whitelist {
      leaf-list voucher-identifier {
        type string;
        description
          "A fingerprint over the voucher artifact.";
      }
      description
        "Indicates that the listed of vouchers are known
        to be good. If a voucher is not listed, then
        it is considered revoked.";
    }

    container blacklist {
      leaf-list voucher-identifier {
        type string;
        description
          "A fingerprint over the voucher artifact.
          Missing if list is empty.";
      }
      description
        "Indicates that the list of vouchers have been
        revoked. If a voucher is not listed, then it
        is considered good.";
    }
  } // end list-type
} // end issuer-wide

container voucher-specific {
  description
    "This revocation provides voucher-specific revocation
    status (similar to an OCSP response).";
```

```
leaf voucher-identifier {
    type string;
    mandatory true;
    description
        "A fingerprint over the voucher artifact.";
}

leaf voucher-status {
    type enumeration {
        enum good {
            description
                "Indicates that this voucher is valid";
        }
        enum revoked {
            description
                "Indicates that this voucher is invalid";
        }
        enum unknown {
            description
                "Indicates that the voucher's status is unknown";
        }
    }
    mandatory true;
    description
        "Indicates if the revocation status for the specified
        voucher.";
}

container revocation-information {
    must "../voucher-status = 'revoked'";

    leaf revoked-on {
        type yang:date-and-time;
        mandatory true;
        description
            "The date this voucher was revoked";
    }

    leaf revocation-reason {
        type enumeration {
            enum unspecified {
                description
                    "Indicates that the reason the voucher
                    was revoked is unspecified.";
            }
            enum key-compromise {
                description
                    "Indicates that the reason the voucher
```

```
        was revoked is because its key was
        compromised.";
    }
    enum issuer-compromise {
        description
        "Indicates that the reason the voucher
        was revoked is because its issuer was
        compromised.";
    }
    enum affiliation-changed {
        description
        "Indicates that the reason the voucher
        was revoked is because its affiliation
        changed (e.g., device assigned to a
        new owner.";
    }
    enum superseded {
        description
        "Indicates that the reason the voucher
        was revoked is because it has been
        superseded (e.g., the previous voucher
        expired.";
    }
    enum cessation-of-operation {
        description
        "Indicates that the reason the voucher
        was revoked is because its issuer has
        ceased operations.";
    }
} // end enumeration

mandatory true;
description
    "modeled after 'CRLReason' in RFC 5280.";
} // end revocation reason

description
    "Provides details regarding why a voucher's revocation.
    Modeled after 'ResponseData' in RFC6960.";

} // end revocation-information

} // end voucher-specific
}

anydata additional-data {
    description
        "Additional data signed by the manufacturer. The manufacturer
```



```
    might put additional data into its voucher revocations, for
    human or device consumption.";

    // Ed. is the additional data normative? - if so, should we
    // remove this free-form field, and assume it will be formally
    // extended later? Note: the zerotouch draft doesn't need this
    // field...
  }
}
```

6. Security Considerations

6.1. Clock Sensitivity

This document defines artifacts containing time values for voucher expirations and revocations, which require an accurate clock in order to be processed correctly. Implementations **MUST** ensure devices have an accurate clock when shipped from manufacturing facilities, and take steps to prevent clock tampering.

If it is not possible to ensure clock accuracy, it is **RECOMMENDED** that implementations disable the aspects of the solution having clock sensitivity. In particular, such implementations should assume that vouchers neither ever expire or are revokable.

It is important to note that implementations **SHOULD NOT** rely on NTP for time, as it is not a secure protocol.

7. IANA Considerations

7.1. The IETF XML Registry

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

```
URI: urn:ietf:params:xml:ns:yang:ietf-voucher
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

```
URI: urn:ietf:params:xml:ns:yang:ietf-voucher-revocation
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

7.2. The YANG Module Names Registry

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

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

name:      ietf-voucher-revocation
namespace: urn:ietf:params:xml:ns:yang:ietf-voucher-revocation
prefix:    vchr
reference:  RFC XXXX
```

8. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name):

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

9.2. Informative References

- [draft-ietf-anima-bootstrapping-keyinfra] Pritikin, M., Richardson, M., Behringer, M., and S. Bjarnason, "Bootstrapping Key Infrastructures", draft-ietf-anima-bootstrapping-keyinfra (work in progress), 2016, <<https://tools.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra>>.

[draft-ietf-netconf-zerotouch]

Watsen, K. and M. Abrahamsson, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch (work in progress), 2016, <<https://tools.ietf.org/html/draft-ietf-netconf-zerotouch>>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.

Appendix A. Change Log

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Michael C. Richardson
Sandelman Software Works

EMail: mcr+ietf@sandelman.ca
URI: <http://www.sandelman.ca/>

Max Pritikin
Cisco Systems

EMail: pritikin@cisco.com

Toerless Eckert
Cisco Systems

EMail: tte+anima@cs.fau.de

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2017

M. Bjorklund, Ed.
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper
R. Wilton
Cisco
October 27, 2016

A Revised Conceptual Model for YANG Datastores
draft-nmdsdt-netmod-revised-datastores-00

Abstract

Datastores are a fundamental concept binding the YANG data modeling language to protocols transporting data defined in YANG data models, such as NETCONF or RESTCONF. This document defines a revised conceptual model of datastores based on the experience gained with the initial simpler model and addressing requirements that were not well supported in the initial model.

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 April 30, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Introduction	2
2. Background	3
3. Terminology	4
4. Original Model of Datastores	4
5. Revised Model of Datastores	6
5.1. The <intended> datastore	8
5.2. The <applied> datastore	8
5.2.1. Missing Resources	9
5.2.2. System-controlled Resources	9
5.3. The <operational-state> datastore	9
6. Implications	9
6.1. Implications on NETCONF	9
6.1.1. Migration Path	10
6.2. Implications on RESTCONF	10
6.3. Implications on YANG	11
6.4. Implications on Data Models	11
7. Data Model Design Guidelines	11
7.1. Auto-configured or Auto-negotiated Values	11
8. Data Model	12
9. IANA Considerations	14
10. Security Considerations	14
11. Acknowledgments	14
12. References	15
12.1. Normative References	15
12.2. Informative References	15
Appendix A. Example Data	16
Appendix B. Open Issues	19
Authors' Addresses	20

1. Introduction

This document provides a revised architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [I-D.ietf-netconf-restconf] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding management data models to network management protocols and agreement on a common architectural model of datastores ensures that data models can be written in a network management

protocol agnostic way. This architectural framework identifies a set of conceptual datastores but it does not mandate that all network management protocols expose all these conceptual datastores. Furthermore, the architecture does not detail how data is encoded by network management protocols.

2. Background

NETCONF [RFC6241] provides the following definitions:

- o datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.
- o configuration datastore: The datastore holding the complete set of configuration data that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [RFC7950] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG did exist):

- o datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.
- o configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration data.

RFC 6244 defined operational state data as follows:

- o Operational state data is a set of data that has been obtained by the system at runtime and influences the system's behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

Section 4.3.3 of RFC 6244 discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as a separate data tree is the recommended approach.

Implementation experience and requests from operators [I-D.ietf-netmod-opstate-reqs], [I-D.openconfig-netmod-opstate] indicate that the datastore model initially designed for NETCONF and refined by YANG needs to be extended. In particular, the notion of intended configuration and applied configuration has developed.

Furthermore, separating operational state data from configuration data in a separate branch in the data model has been found operationally complicated. The relationship between the branches is not machine readable and filter expressions operating on configuration data and on related operational state data are different.

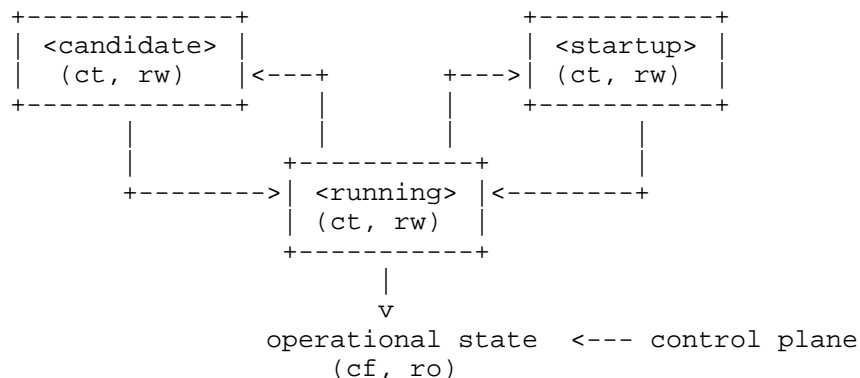
3. Terminology

This document defines the following terms:

- o configuration data: Data that determines how a device behaves. Configuration data can originate from different sources. In YANG 1.1, configuration data is the "config true" nodes.
- o static configuration data: Configuration data that is eventually persistent and used to get a device from its initial default state into its desired operational state.
- o dynamic configuration data: Configuration data that is obtained dynamically during the operation of a device through interaction with other systems and not persistent.
- o system configuration data: Configuration data that is supplied by the device itself.
- o data-model-defined configuration data: Configuration data that is not explicitly provided but for which a value defined in the data model is used. In YANG 1.1, such data can be defined with the "default" statement or in "description" statements.

4. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:



ct = config true; cf = config false
 rw = read-write; ro = read-only
 boxes denote datastores

Note that read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for the <candidate> and <startup> datastores is optional and the <running> datastore does not have to be writable. Furthermore, the <startup> datastore can only be modified by copying <running> to <startup> in the standardized NETCONF datastore editing model. The RESTCONF protocol does not expose these differences and instead provides only a writable unified datastore, which hides whether edits are done through a <candidate> datastore or by directly modifying the <running> datastore or via some other implementation specific mechanism. RESTCONF also hides how configuration is made persistent. Note that implementations may also have additional datastores that can propagate changes to the <running> datastore. NETCONF explicitly mentions so called named datastores.

Some observations:

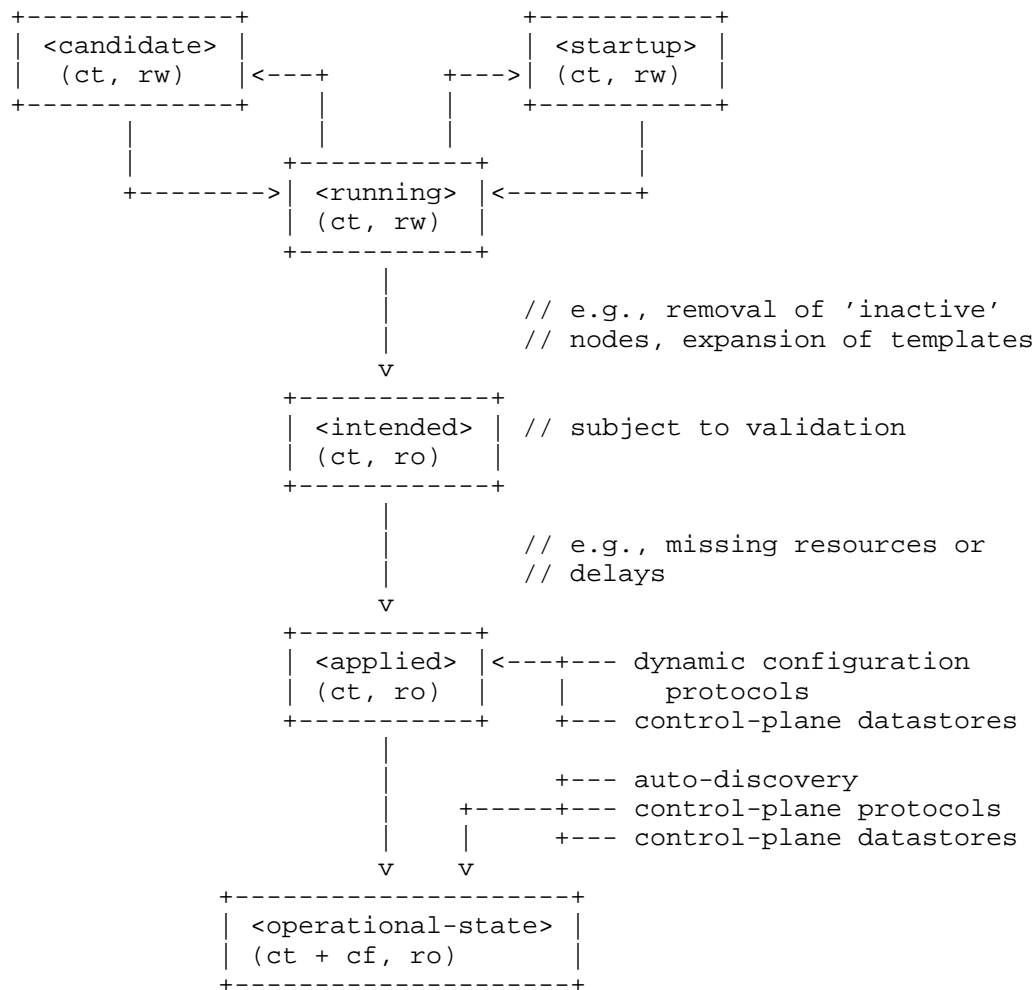
- o Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.
- o The NETCONF <get/> operation returns the content of the <running> configuration datastore together with the operational state. It is therefore necessary that config false data is in a different branch than the config true data if the operational state data can have a different lifetime compared to configuration data or if configuration data is not immediately or successfully applied.
- o Several implementations have proprietary mechanisms that allow clients to store inactive data in the <running> datastore; this

inactive data is only exposed to clients that indicate that they support the concept of inactive data; clients not indicating support for inactive data receive the content of the <running> datastore with the inactive data removed. Inactive data is conceptually removed during validation.

- o Some implementations have proprietary mechanisms that allow clients to define configuration templates in <running>. These templates are expanded automatically by the system, and the resulting configuration is applied internally.
- o Some operators have reported that it is essential for them to be able to retrieve the configuration that has actually been successfully applied, which may be a subset or a superset of the <running> configuration.

5. Revised Model of Datastores

Below is a new conceptual model of datastores extending the original model in order to reflect the experience gained with the original model.



ct = config true; cf = config false
 rw = read-write; ro = read-only
 boxes denote datastores

The model foresees control-plane datastores that are by definition not part of the persistent configuration of a device. In some contexts, these have been termed ephemeral datastores since the information is ephemeral, i.e., lost upon reboot. The control-plane datastores interact with the rest of the system through the <applied> or <operational-state> datastores, depending on the type of data they contain. Note that the ephemeral datastore discussed in I2RS documents maps to a control-plane datastore in the revised datastore model described here.

5.1. The <intended> datastore

The <intended> datastore is a read-only datastore that consists of config true nodes. It is tightly coupled to <running>. When data is written to <running>, the data that is to be validated is also conceptually written to <intended>. Validation is performed on the contents of <intended>.

On a traditional NETCONF implementation, <running> and <intended> are always the same.

Currently there are no standard mechanisms defined that affect <intended> so that it would have different contents than <running>, but this architecture allows for such mechanisms to be defined.

One example of such a mechanism is support for marking nodes as inactive in <running>. Inactive nodes are not copied to <intended>, and are thus not taken into account when validating the configuration.

Another example is support for templates. Templates are expanded when copied into <intended>, and the result is validated.

5.2. The <applied> datastore

The <applied> datastore is a read-only datastore that consists of config true nodes. It contains the currently active configuration on the device. This data can come from several sources; from <intended>, from dynamic configuration protocols (e.g., DHCP), or from control-plane datastores.

As data flows into the <applied> and <operational-state> datastores, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The "origin" metadata annotation is defined in Section 8. The values are YANG identities. The following identities are defined:

```
+-- origin
  +-- static
  +-- dynamic
  +-- data-model
  +-- system
```

These identities can be further refined, e.g., there might be an identity "dhcp" derived from "dynamic".

The <applied> datastore contains the subset of the instances in the <operational-state> datastore where the "origin" values are derived from or equal to "static" or "dynamic".

5.2.1. Missing Resources

Sometimes some parts of <intended> configuration refer to resources that are not present and hence parts of the <intended> configuration cannot be applied. A typical example is an interface configuration that refers to an interface that is not currently present. In such a situation, the interface configuration remains in <intended> but the interface configuration will not appear in <applied>.

5.2.2. System-controlled Resources

Sometimes resources are controlled by the device and such system controlled resources appear in (and disappear from) the <operational-state> dynamically. If a system controlled resource has matching configuration in <intended> when it appears, the system will try to apply the configuration, which causes the configuration to appear in <applied> eventually (if application of the configuration was successful).

5.3. The <operational-state> datastore

The <operational-state> datastore is a read-only datastore that consists of config true and config false nodes. In the original NETCONF model the operational state only had config false nodes. The reason for incorporating config true nodes here is to be able to expose all operational settings without having to replicate definitions in the data models.

The <operational-state> datastore contains all configura data actually used by the system, i.e., all applied configuration, system configuration and data-model-defined configuration. This data is marked with the "origin" metadata annotation. In addition, the <operational-state> datastore also contains state data.

In the <operational-state> datastore, semantic constraints defined in the data model are not applied. See Appendix B.

6. Implications

6.1. Implications on NETCONF

- o A mechanism is needed to announce support for <intended>, <applied>, and <operational-state>.

- o Support for `<intended>`, `<applied>`, and `<operational-state>` should be optional to implement.
- o For systems supporting `<intended>` or `<applied>` configuration datastores, the `<get-config/>` operation may be used to retrieve data stored in these new datastores.
- o A new operation should be added to retrieve the operational state data store (e.g., `<get-state/>`). An alternative is to define a new operation to retrieve data from any datastore (e.g., `<get-data>` with the name of the datastore as a parameter). In principle `<get-config/>` could work but it would be a confusing name.
- o The `<get/>` operation will be deprecated since it returns data from two datastores that may overlap in the revised datastore model.

6.1.1. Migration Path

A common approach in current data models is to have two separate trees `"/foo"` and `"/foo-state"`, where the former contains config true nodes, and the latter config false nodes. A data model that is designed for the revised architectural framework presented in this document will have a single tree `"/foo"` with a combination of config true and config false nodes.

A server that implements the `<operational-state>` datastore can implement a module of the old design. In this case, some instances are probably reported both in the `"/foo"` tree and in the `"/foo-state"` tree.

A server that does not implement the `<operational-state>` datastore can implement a module of the new design, but with limited functionality. Specifically, it may not be possible to retrieve all operationally used instances (e.g., dynamically configured or system-controlled). The same limitation applies to a client that does not implement the `<operational-state>` datastore, but talks to a server that implements it.

6.2. Implications on RESTCONF

- o The `{+restconf}/data` resource represents the combined configuration and state data resources that can be accessed by a client. This is effectively bundling `<running>` together with `<operational-state>`, much like the `<get/>` operation of NETCONF. This design should be deprecated.

- o A new query parameter is needed to indicate that data from `<operational-state>` is requested.

6.3. Implications on YANG

- o Some clarifications may be needed if this revised model is adopted. YANG currently describes validation in terms of the `<running>` configuration datastore while it really happens on the `<intended>` configuration datastore.

6.4. Implications on Data Models

- o Since the NETCONF `<get/>` operation returns the content of the `<running>` configuration datastore and the operational state together in one tree, data models were often forced to branch at the top-level into a config true branch and a structurally similar config false branch that replicated some of the config true nodes and added state nodes. With the revised datastore model this is not needed anymore since the different datastores handle the different lifetimes of data objects. Introducing this model together with the deprecation of the `<get/>` operation makes it possible to write simpler models.
- o There may be some differences in the value set of some nodes that are used for both configuration and state. At this point of time, these are considered to be rare cases that can be dealt with using different nodes for the configured and state values.
- o It is important to design data models with clear semantics that work equally well for instantiation in a configuration datastore and instantiation in the `<operational-state>` datastore.

7. Data Model Design Guidelines

7.1. Auto-configured or Auto-negotiated Values

Sometimes configuration leafs support special values that instruct the system to automatically configure a value. An example is an MTU that is configured to 'auto' to let the system determine a suitable MTU value. Another example is Ethernet auto-negotiation of link speed. In such a situation, it is recommended to model this as two separate leafs, one config true leaf for the input to the auto-negotiation process, and one config false leaf for the output from the process.

8. Data Model

```
<CODE BEGINS> file "ietf-yang-architecture@2016-10-13.yang"

module ietf-yang-architecture {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-architecture";
  prefix arch;

  import ietf-yang-metadata {
    prefix md;
  }

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

  contact
    "WG Web:  <https://datatracker.ietf.org/wg/netmod/>

    WG List:  <mailto:netmod@ietf.org>

    Editor:   Martin Bjorklund
              <mailto:mbj@tail-f.com>";

  description
    "This YANG module defines an 'origin' metadata annotation,
    and a set of identities for the origin value.  The 'origin'
    metadata annotation is used to mark data in the applied
    and operational state datastores with information on where
    the data originated.

    Copyright (c) 2016 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

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

    This version of this YANG module is part of RFC XXXX
    (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
    for full legal notices.";

  revision 2016-10-13 {
    description
      "Initial revision.";
    reference
```



```
    "RFC XXXX: A Revised Conceptual Model for YANG Datastores";
}

/*
 * Identities
 */

identity origin {
    description
        "Abstract base identity for the origin annotation.";
}

identity static {
    base origin;
    description
        "Denotes data from static configuration (e.g., <intended>).";
}

identity dynamic {
    base origin;
    description
        "Denotes data from dynamic configuration protocols
        or dynamic datastores (e.g., DHCP).";
}

identity system {
    base origin;
    description
        "Denotes data created by the system independently of what
        has been configured.";
}

identity data-model {
    base origin;
    description
        "Denotes data that does not have an explicitly configured
        value, but has a default value in use.  Covers both simple
        defaults and complex defaults.";
}

/*
 * Metadata annotations
 */

md:annotation origin {
    type identityref {
        base origin;
    }
}
```

```
    }  
  }  
  
<CODE ENDS>
```

9. IANA Considerations

TBD

10. Security Considerations

This document discusses a conceptual model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

11. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([I-D.bjorklund-netmod-operational], [I-D.wilton-netmod-opstate-yang], [I-D.ietf-netmod-opstate-reqs], [I-D.kwatsen-netmod-opstate], [I-D.openconfig-netmod-opstate]) and [RFC6244] touched on some of the problems of the original datastore model. The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

- o Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- o Andy Bierman, YumaWorks, <andy@yumaworks.com>
- o Marcus Hines, Google, <hines@google.com>
- o Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- o Acee Lindem, Cisco Systems, <acee@cisco.com>
- o Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- o Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- o Anees Shaikh, Google, <aashaikh@google.com>
- o Rob Shakir, Google, <robjs@google.com>

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

12. References

12.1. Normative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-17 (work in progress), September 2016.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <<http://www.rfc-editor.org/info/rfc7952>>.

12.2. Informative References

- [I-D.bjorklund-netmod-operational]
Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.
- [I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.
- [I-D.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.
- [I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

- [I-D.wilton-netmod-opstate-yang]
Wilton, R., "With-config-state" Capability for NETCONF/
RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in
progress), December 2015.
- [RFC6244] Shafer, P., "An Architecture for Network Management Using
NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June
2011, <<http://www.rfc-editor.org/info/rfc6244>>.

Appendix A. Example Data

In this example, the following fictional module is used:

```
module example-system {
  yang-version 1.1;
  namespace urn:example:system;
  prefix sys;

  import ietf-inet-types {
    prefix inet;
  }

  container system {
    leaf hostname {
      type string;
    }
  }

  list interface {
    key name;

    leaf name {
      type string;
    }

    container auto-negotiation {
      leaf enabled {
        type boolean;
        default true;
      }
      leaf speed {
        type uint32;
        units mbps;
        description
          "The advertised speed, in mbps.";
      }
    }
  }

  leaf speed {
```

```

        type uint32;
        units mbps;
        config false;
        description
            "The speed of the interface, in mbps.";
    }

    list address {
        key ip;

        leaf ip {
            type inet:ip-address;
        }
        leaf prefix-length {
            type uint8;
        }
    }
}
}
}
}

```

The operator has configured the host name and two interfaces, so the contents of <intended> is:

```

<system xmlns="urn:example:system">

  <hostname>foo</hostname>

  <interface>
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

  <interface>
    <name>eth1</name>
    <address>
      <ip>2001:db8::20</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

</system>

```

The system has detected that the hardware for one of the configured interfaces ("eth1") is not yet present, so the configuration for that interface is not applied. Further, the system has received a host name and an additional IP address for "eth0" over DHCP. This is reflected in <applied>:

```
<system
  xmlns="urn:example:system"
  xmlns:arch="urn:ietf:params:xml:ns:yang:ietf-yang-architecture">

  <hostname arch:origin="arch:dynamic">bar</hostname>

  <interface arch:origin="arch:static">
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>32</prefix-length>
    </address>
    <address arch:origin="arch:dynamic">
      <ip>2001:db8::1:100</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

</system>
```

In <operational-state>, all data from <applied> is present, in addition to a default value, a loopback interface automatically added by the system, and the result of the "speed" auto-negotiation:

```
<system
  xmlns="urn:example:system"
  xmlns:arch="urn:ietf:params:xml:ns:yang:ietf-yang-architecture">

  <hostname arch:origin="arch:dynamic">bar</hostname>

  <interface arch:origin="arch:static">
    <name>eth0</name>
    <auto-negotiation>
      <enabled arch:origin="arch:data-model">true</enabled>
      <speed>1000</speed>
    </auto-negotiation>
    <speed>100</speed>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>32</prefix-length>
    </address>
    <address arch:origin="arch:dynamic">
      <ip>2001:db8::1:100</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

  <interface arch:origin="arch:system">
    <name>lo0</name>
    <address>
      <ip>::1</ip>
      <prefix-length>128</prefix-length>
    </address>
  </interface>

</system>
```

Appendix B. Open Issues

1. Do we need another DS `<active>` inbetween `<running>` and `<intended>`? This DS would allow a client to see all active nodes, including unexpanded templates.
2. How do we handle semantical constraints in `<operational-state>`? Are they just ignored? Do we need a new YANG statement to define if a "must" constraints applies to the `<operational-state>`?
3. Should it be possible to ask for `<applied>` in RESTCONF?
4. Better name for "static configuration"?
5. Better name for "intended"?

Authors' Addresses

Martin Bjorklund (editor)
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper

Email: phil@juniper.net

Kent Watsen
Juniper

Email: kwatsen@juniper.net

Rob Wilton
Cisco

Email: rwilton@cisco.com