This document defines a subscription and push mechanism for YANG datastores. This mechanism allows client applications to request updates from a YANG datastore, which are then pushed by the server to the client per a subscription policy, without requiring additional client 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 September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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
1. Introduction

YANG [RFC6020] was originally designed for the Netconf protocol [RFC6241], which originally put most emphasis on configuration. However, YANG is not restricted to configuration data. YANG datastores, i.e. datastores that contain data modeled according using YANG, can contain configuration as well as contain operational data. It is therefore reasonable to expect that data in YANG datastores will increasingly be used to support applications that are not
focused on managing configurations but that are, for example, related to service assurance.

Service assurance applications typically involve monitoring operational state of networks and devices; of particular interest are changes that this data undergoes over time. Likewise, there are applications in which data and objects from one datastore need to be made available both to applications in other systems and to remote datastores [peermount-req][peermount]. This requires mechanisms that allow remote systems to become quickly aware of any updates to allow to validate and maintain cross-network integrity and consistency.

Traditional approaches rely heavily on polling, in which data is periodically explicitly retrieved by a client from a server to stay up-to-date.

There are various issues associated with polling-based management:

- It introduces additional load on network and devices. Each polling cycle requires a separate yet arguably redundant request that results in an interrupt, requires parsing, consumes bandwidth.
- 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.
- 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.

More effective is an alternative in which an application can request to be automatically updated of current content of the datastore (such as a subtree, or data in a subtree that meets a certain filter condition), and in which the server that maintains the datastore subsequently pushes those updates. However, such a solution does not currently exist.

The need to perform polling-based management is typically considered an important shortcoming of management 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 using protocols such as NETCONF [RFC6241] or RESTconf [restconf] will be
any more effective, as they would follow the same request/response pattern.

While YANG allows the definition of notifications, such notifications are generally intended to indicate the occurrence of certain well-specified event conditions, such as a the onset of an alarm condition or the occurrence of an error. A capability to subscribe to and deliver 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.

Accordingly, there is a need for a service that allows client applications to subscribe to updates of a YANG datastore and that allows the server to push those updates. The requirements for such a service are documented in [pub-sub-reqs]. This document proposes a solution that addresses those requirements. The solution features the following:

- A mechanism that allows clients to subscribe to automatic datastore updates, and the means to manage those subscription. The subscription allows clients to specify which data they are interested in, and to provide optional filters with criteria that data must meet for updates to be sent. Furthermore, subscription 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.

- The ability to negotiate subscription parameters. Because not every server may support every requested interval for every piece of data, it is necessary for a server to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate subscription parameters.

- A mechanism is to communicate the updates themselves. For this, the proposal leverages and extends existing YANG/Netconf/Restconf mechanisms, defining special notifications that carry updates.

This document specifies a YANG data model to manage subscriptions to data in YANG datastores, and to configure associated filters and data streams. It defines extensions to RPCs defined in [RFC5277] that allow to extend notification subscriptions to subscriptions for datastore updates. It also defines a notification that can be used to carry data updates and thus serve as push mechanism.
2. Definitions and Acronyms

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

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.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

Push-update stream: A conceptual data stream of a datastore that streams the entire datastore contents continuously and perpetually.

RPC: Remote Procedure Call

SNMP: Simple Network Management Protocol

Subscription: A contract between a client ("subscriber") and a server ("publisher"), stipulating which information the client wishes to receive from the server (and which information the server has to provide to the client) without the need for further solicitation.

Subscription 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.

Subscription policy: A policy that specifies under what circumstances to push an update, e.g., whether updates are to be provided periodically or only whenever changes occur.

Update: A data item containing the current value of a data node.

Update trigger: A trigger, as specified by a subscription policy, that causes an update to be sent, respectively a data record to be generated. An example of a trigger is 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.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Solution Overview

This document specifies a solution that allows clients to subscribe
to information updates in a YANG datastore, which are subsequently
pushed from the server to the client.

Subscriptions are initiated by clients. Servers respond to a
subscription request explicitly positively or negatively. Negative
responses include information about why the subscription was not
accepted, in order to facilitate converging on an acceptable set of
subscription parameters. Once a subscription has been established,
datastore push updates are pushed from the server to the subscribing
client until the subscription ends.

Accordingly, the solution encompasses several components:

- The subscription model for configuration and management of the
  subscriptions, with a set of associated services.
- The ability to negotiate subscription parameters, in cases where a
  subscription desired by a client cannot currently be served.
- The stream of datastore push updates.

In addition, there are a number of additional considerations, such as
the tie-in of the mechanisms with security mechanisms. Each of those
aspects will be discussed in the following subsections.

3.1. Subscription Model

Yang-push subscriptions are defined using a data model. This model
is based on the subscriptions defined in [RFC-5277], which is also
reused in RESTconf. The model is extended with several parameters,
including a subscription type and a subscription ID.

A subscription refers to a datastream. The subscription model
assumes the presence of a conceptual perpetual datastream "push-
update" of continuous datastore updates of infinite time resolution.
A subscription refers to this datastream and specifies filters that
are to be applied to, it for example, to provide only those subsets
of the information that match a filter criteria. In addition, a
subscription specifies a subscription policy that defines the trigger when data records should be sent, for example at periodic intervals or whenever underlying data items change.

The complete set of subscription parameters is as follows:

- The name of the stream to subscribe to. The subscription model always assumes the presence of a perpetual and continuous stream of updates. The stream is called "push-update". However, as mentioned, it is possible to subscribe to other datastreams, such as custom datastreams which are separately configured.

- Optional filter(s), describing the subset of data items in the stream’s data records that are of interest to the subscriber. The server should only send to the subscriber the data items that match the filter(s), when present. 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. Two filtering mechanisms are provided: subtree filtering and Xpath filtering, with the semantics described in [RFC 5277 Section 3.6]. (Additional filter types can be added through extensions.)

- An identifier for the subscription.

- An optional start time. Used to trigger replays starting at the provided time. Its semantics are those in [RFC 5277].

- An optional stop time. Used to limit temporarily the events of interest. Its semantics are those in [RFC 5277].

- For subscriptions to "push-update", a subscription policy definition regarding the update trigger to send new updates. The trigger can be periodic or based on change. For periodic subscriptions, the trigger is defined by the interval with which to push updates. For on-change subscriptions, the trigger is defined using the dampening interval with which to push repeated changes, an indicator for the magnitude of changes, etc.

It is conceivable that additional subscription parameters might be added in the future. For example, a parameter might be introduced that allows to specify which encoding method for updates to use. The list could accordingly be extended and augmented in the future.
3.2. Negotiation of Subscription Policies

A subscription rejection can be caused by the inability of the server to provide a stream with the requested semantics. Providing "on-change" updates for operational data can be computationally expensive and an agent may decide not to support them or supporting them for a small number of subscribers or for a limited set of data nodes.

Yang-push supports a simple negotiation between clients and servers for subscription parameters. The negotiation is limited to a single pair of subscription request and response. For negative responses, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters are no guarantee for subsequent requests for this client or others.

3.3. Custom Datastreams

Optionally, it is possible to introduce other datastreams (beyond the datastore-push datastream) with custom semantics. Some datastreams can be custom configured. The support of this is tied to a separate feature. The configuration of a custom datastream specifies the trigger conditions under which new data records for the stream are generated, and which updates the corresponding data records contain. For example, the configuration of a datastream can specify which subsets of data nodes in a datastore the datastream should contain, which filter criteria the updates need to meet, and under what conditions to create updates – for example, periodically or whenever a data item changes.

A subscription that refers to a custom datastream can specify a set of filters, like for the "push-update" datastream. However, the policy as to when updates are triggered (periodically or on change) needs to be the same as the policy of the datastream and cannot be modified. It is not possible, for example, to define a custom datastream which creates on-change updates, yet subscribe to that datastream with periodic updates.

While conceptually similar, the choice between subscribing to datastream "push-update" or configuring and subscribing to a custom datastream can be thought of as analogous to the choice between operating a nozzle that is connected to a hose, or controlling the faucet (custom datastream). Operating the nozzle is for most uses simpler; however, the option to operate the faucet instead can provide additional flexibility in some scenarios.
3.4. 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.

A more suitable mechanism is therefore that of a notification. Contrary to notifications associated with alarms and unexpected event occurrences, push updates are solicited, i.e. tied tied to a particular subscription which triggered the notification. (An alternative conceptual model would consider a subscription an "opt-in" filter on a continuous stream of updates.)

The notification contains several parameters:

- A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.
- A data node that contains a representation of the datastore subtree containing the updates. The subtree is filtered per access control rules to contain only data that the subscriber is authorized to see. Also, depending on the subscription type, i.e., specifically for on-change subscriptions, the subtree contains only the data nodes that contain actual changes. (This can be simply a node of type string or, for XML-based encoding, anyxml.)

Notifications are sent using <notification> elements as defined in [RFC5277]. Alternative transports are conceivable but outside the scope of this specification.

The solution specified in this document uses notifications to communicate datastore updates. The contents of the notification includes a set of explicitly defined data nodes. For this purpose, a new generic notification is introduced, "push-update" notification. This notification is used to carry a data record with updates of datastore contents as specified by a subscription.

The update record consists of a data snippet that contains an instantiated datastore subtree with the subscribed contents. Data nodes that do not match filter criteria are removed. Likewise, in the case of a subscription with "on-change" subscription policy, data nodes that have not undergone change are omitted. The contents of the update record 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 server, excluding the following statements: "mandatory", "must", "min-elements", "max-elements", "when", and "default". However, in a YANG data model, it is not practical to model the precise data contained in the updates as part of the notification, because the specific data nodes supported depends 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.

The following is an example of push notification. It contains an update for subscription my-sub, including a subtree with root foo that contains a leaf, bar:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <subscription-id xmlns="urn:ietf:params:xml:ns:netconf:subscription-push:1.0">
    my-sub
  </subscription-id>
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <datastore-contents xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
    <foo>
      <bar>some_string</bar>
    </foo>
  </datastore-contents>
</notification>
```

Figure 1: Push example

3.5. Subscription operations

There are several operations associated with subscriptions. At the most basic level, clients need to be able to create subscriptions, as well as delete subscriptions when they are no longer needed.

RFC 5277 specifies an operation to create subscriptions for event streams, `<create-subscription>`. This operation is leveraged and extended to create datastore-push subscriptions. Specifically, an additional parameter is added to allow for the specification of trigger policy.

To support datastore push, a server MUST support the interleave capability specified in [RFC5277]. This is required to allow for
modification of what data is being subscribed to without needing to establish a separate Netconf session.

The example below illustrates a subscription for a periodic push of all data under a container called foo.

```
<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">
  <stream>push-update</stream>
  <subscription-id xmlns="urn:ietf:params:xml:ns:netconf:
datastore-push:1.0">
   my-sub
  </subscription-id>
  <filter netconf:type="xpath"
   xmlns:ex="http://example.com/foo/1.0"
   select="/ex:foo"/>
  <period xmlns="urn:ietf:params:xml:ns:netconf:datatype:
datastore-push:1.0">500</period>
 </create-subscription>
</netconf:rpc>
```

Figure 2: Subscription example

The example below illustrates a subscription response, where an agent does not support frequent periodic updates, and suggests a different sampling rate to the client.

```
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
  <error-tag>operation-not-supported</error-tag>
  <error-severity>error</error-severity>
  <error-info>
   <supported-subscription xmlns="urn:ietf:params:xml:ns:netconf:
datastore-push:1.0">
    <period>3000</period>
   </supported-subscription>
  </error-info>
 </rpc-error>
</rpc-reply>
```

Figure 3: Subscription negotiation example

RFC 5277 does not specify operations to delete subscriptions. Instead, it assumes that an event subscription is associated with its
own Netconf session. When the session is torn down, the subscription is implicitly deleted. Likewise, there is no operation to modify a subscription. Modifying a subscription requires tearing down a Netconf session, starting a new one, and creating a new subscription. Furthermore, each session only supports a single subscription. Establishing multiple subscriptions requires multiple concurrent Netconf sessions.

To facilitate datastore-push subscriptions, an additional RPC is introduced, `<delete-subscription>`.

The `<delete-subscription>` operation takes as parameter a subscription ID. As a result of the operation, the subscription is removed and no more data records will be sent.

```xml
<netconf:rpc message-id="102"
 xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <subscription-id xmlns="urn:ietf:params:xml:ns:netconf:
datastore-push:1.0">
      my-sub
    </subscription-id>
  </delete-subscription>
</netconf:rpc>
```

Figure 4: Subscription deletion

Finally, a separate operation to modify a subscription is introduced, `<modify-subscription>`.

This operation takes the same parameters as `<create-subscription>`, but refers to an existing subscription. Of course, a subscription could also be deleted and another be created. However, modify operation avoids issues regarding the synchronization of creation and deletion operations, such as potential loss or duplication of updates. Also, a modify operation allows to simply extend an existing subscription beyond the initial subscription end time.
3.6. A YANG data model for management of datastore push subscriptions

Subscriptions as well as datastreams can be subjected to management themselves. For example, it is possible that a server 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 server needs to be able to temporarily suspend the subscription, or even to terminate it. More generally, the server should provide a means by which the status of subscriptions can be monitored. When custom datastreams are supported, those datastreams need to be configured and monitored as well.

For this purpose, a YANG data model is introduced, which is depicted in the following figure.
module: ietf-datastore-push
  +--rw streams {custom-streams}?
    +--rw stream* [stream-name]
      +--rw stream-name         string
      +--ro stream-status?      identityref
      +--rw subtree-filter?     subtree-filter
      +--rw xpath-filter?       yang:xpath1.0
      +--rw (update-trigger)?
        +--:(periodic)
        |  +--rw period?             yang:timeticks
        +--:(on-change)
        |  +--rw dampening-period    yang:timeticks
        +--rw (change-policy)?
          +--:(delta-policy)
            +--rw delta?              uint32

  +--rw subscriptions
    +--ro datastore-push-subscription* [subscription-id]
      +--ro subscription-id        subscription-identifier
      +--ro subscription-status?   identityref
      +--ro stream?                string
      +--ro start-time?            yang:date-and-time
      +--ro stop-time?             yang:date-and-time
      +--ro subtree-filter?        subtree-filter
      +--ro xpath-filter?          yang:xpath1.0
      +--ro (update-trigger)?
        +--:(periodic)
        |  +--ro period?             yang:timeticks
        +--:(on-change)
        |  +--ro dampening-period    yang:timeticks
        +--ro (change-policy)?
          +--:(delta-policy)
            +--ro delta?              uint32

Figure 6: Model structure

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. The subscription status indicates whether
the subscription is currently active and healthy, or if it is
degraded in some form. Subscriptions are automatically removed from
the list once they expire or are terminated. Because subscriptions
are managed using their own set of operation primitives, they are
read-only.

An optional feature, custom-streams, is introduced to allow for the
configuration of custom datastreams. Custom datastreams are
represented through a separate list, consisting of information used
to configure those datastreams. This information constitutes mostly configuration information, with the exception of parameters used to indicate the status and health of the datastream.

In addition, a server needs to indicate any changes in status to the subscriber through a notification. Specifically, subscribers need to be informed of the following:

- A subscription has been temporarily suspended (including the reason)
- A subscription (that had been suspended earlier) is once again operational
- A subscription has been abnormally terminated (including the reason)
- A subscription has been modified (including the current set of subscription parameters in effect)

Finally, a server might 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.

3.7. Other considerations

3.7.1. Authorization

A client may only receive updates to data that the client has proper authorization for. Normal authorization rules apply. Data that is being pushed therefore needs to be subjected to a filter that applies the corresponding rules, removing any non-authorized data as applicable.

The authorization model for data in YANG datastores is described in the Netconf Access Control Model [RFC6536].

3.7.2. Additional subscription primitives

Other possible operations include the ability to suspend and resume subscriptions. However, those operations are not viewed as essential, as it is always possible to alternatively simply remove a subscription and recreate it when needed.
3.7.3. Robustness and reliability considerations

Particularly in the case of on-change push updates, it is important that push updates do not get lost. However, datastore-push uses a secure and reliable transport. Notifications are not getting reordered, and in addition contain a time stamp. For those reasons, we believe that additional reliability mechanisms at the application level, such as sequence numbers for push updates, are not required.

3.7.4. 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 of operational state alone, to be able to push it, can consume considerable system resources. Configuration data may in many cases be persisted in an actual database or a configuration file, where retrieval of the database content or the file itself is reasonably straightforward and computationally inexpensive. However, retrieval of operational 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, if is important for an implementation to understand what subscriptions it can or cannot support. It is far preferrable to decline a subscription request, than to accept it only to result in subsequent failure later.

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.

4. YANG module

<CODE BEGINS>
file "ietf-datastore-push@2014-03-09.yang"

module ietf-datastore-push {
  namespace "urn:XXXX:params:xml:ns:yang:ietf-datastore-push";
  prefix datastore-push;

import ietf-yang-types {
    prefix yang;
}

organization "IETF";
contact
    "Editor:   Alexander Clemm
        <mailto:alex@cisco.com>
    Editor:   Alberto Gonzalez Prieto
        <mailto:albertgo@cisco.com>
    Editor:   Eric Voit
        <mailto:evoit@cisco.com>";
description
    "This module contains conceptual YANG specifications for datastore push.";
revision 2014-03-09 {
    description
        "Initial revision.";
    reference "Datastore push.";
}
feature custom-streams {
    description
        "This feature allows users to configure datastore update streams in addition to the stream provided by default, datastore-push.";
}
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 in-error {
  base subscription-stream-status;
  description
    "The status is in error or degraded, meaning that
    stream and/or subscription are currently unable to provide
    the negotiated updates.";
}

identity subscription-errors {
  description
    "Base identity for subscription errors.";
}

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 other {
  base subscription-errors;
  description
    "Fallback reason - any other reason";
}

typedef datastore-contents {
  type string;
  description
    "This type is be used to represent datastore contents,
    including a filtered datastore subtree per a set of
    subscription parameters.";
}

typedef subtree-filter {
  type string;
  description
    "This type is used to define a subtree filter.
    Its precise syntax is TBD.";
}
typedef subscription-identifier {
  type string {
    length "1 .. max";
  }
  description
    "A client-provided identifier for the subscription.";
}

typedef subscription-term-reason {
  type identityref {
    base subscription-errors;
  }
  description
    "Reason for a server to terminate a subscription.";
}

typedef subscription-susp-reason {
  type identityref {
    base subscription-errors;
  }
  description
    "Reason for a server to suspend a subscription.";
}

grouping subscription-stream-policy {
  description
    "This grouping contains the parameters which describe the policy which data is pushed as part of a subscription or a data stream.";
  leaf subtree-filter {
    description
      "Datastore subtree of interest.";
    type subtree-filter;
  }
  leaf xpath-filter {
    type yang:xpath1.0;
    description
      "Xpath defining the data items of interest.";
  }
  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;
  description
  "Elapsed time between notifications.";
}
}
case 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 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.";
}
choice change-policy {
  description
  "Policy describing necessary conditions for
  sending an event to the subscriber.";
case delta-policy {
  leaf delta {
    type uint32;
    description
    "For integer, minimum difference
    between current and last reports
    values that can trigger an update.";
  }
}
}
}
}
}
grouping subscription-info {
  description
  "This grouping describes basic information concerning a
  subscription, without the subscription policy which is
  defined separately to be shareable with the definition
  of a datastream.";
leaf stream {
  type string;
  description
  "The name of the stream subscribed to.";
}
leaf start-time {
  type yang:date-and-time;
  description
      "Starting time for replays.";
  reference "RFC 5277, Section 2.1.1";
}
leaf stop-time {
  type yang:date-and-time;
  description
      "Time limit for events of interest.";
  reference "RFC 5277, Section 2.1.1";
}

notification push-update {
  description
      "This notification contains an update from a datastore";
  leaf subscription-id {
    type subscription-identifier;
    mandatory true;
    description
      "This references the subscription because of which the
      notification is sent.";
  }
  leaf datastore-contents {
    type datastore-contents;
    description
      "This contains datastore contents
      per the subscription.";
  }
}

notification subscription-suspended {
  description
      "This notification indicates that a suspension of the
      subscription by the server has occurred. No further
      datastore updates will be sent until subscription
      resumes.";
  leaf subscription-id {
    type subscription-identifier;
    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 {
    description
        "This notification indicates that a subscription that had
        previously been suspended has resumed. Datastore updates
        will once again be sent.";
    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description
        "This references the affected subscription.";
    }
}

notification subscription-modified {
    description
        "This notification indicates that a subscription has
        been modified. Datastore updates sent from this point
        on will conform to the modified terms of the
        subscription.";
    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description
        "This references the affected subscription.";
    }
    uses subscription-info;
    uses subscription-stream-policy;
}

notification subscription-terminated {
    description
        "This notification indicates that a subscription has been
        terminated.";
    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description
        "This references the affected subscription.";
    }
    leaf reason {
        type subscription-term-reason;
        description
        "Provides a reason for why the subscription was
        terminated.";
    }
}

container streams {
    if-feature custom-streams;
list stream {
    key "stream-name";
    description 
        "A user-definable stream.";
    leaf stream-name {
        type string;
        mandatory true;
        description 
            "The name assigned to the stream.";
    }
    leaf stream-status {
        type identityref {
            base subscription-stream-status;
        }
        config false;
    }
    uses subscription-stream-policy;
}

container subscriptions {
    list datastore-push-subscription {
        description 
            "Content of a yang-push subscription. Subscriptions are created using a dedicated RPC, hence they do not constitute configuration information.";
        config false;
        key "subscription-id";
        leaf subscription-id {
            type subscription-identifier;
            description 
                "Identifier to use for this subscription.";
        }
        leaf subscription-status {
            type identityref {
                base subscription-stream-status;
            }
            description 
                "The status of the subscription.";
        }
        uses subscription-info;
        uses subscription-stream-policy;
    }
}

<CODE ENDS>
5. Security Considerations

Subscriptions could be used to attempt to overload servers of YANG datastores. For this reason, it is important that the server has the ability to decline a subscription request if it would deplete its resources. In addition, a server needs to be able to suspend an existing subscription when needed. When this occurs, the subscription status is updated accordingly and the clients are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a client has not 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 client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

6. References

6.1. Normative References


6.2. Informative References

[peermount]
Clemm, A., Medved, J., and E. Voit, "Mounting YANG-defined information from remote datastores", draft-clemm-netmod-mount-02 (work in progress), October 2014.

[peermount-req]
Voit, E., Clemm, A., Bansal, S., Tripathy, A., and P. Yellai, "Requirements for Peer Mounting of YANG subtrees from Remote Datastores", draft-voit-netmod-peer-mount-requirements-00 (work in progress), September 2014.

[pub-sub-reqs]
Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-00 (work in progress), March 2015.

[restconf]

Authors' Addresses

Alexander Clemm
Cisco Systems
EMail: alex@cisco.com

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

Eric Voit
Cisco Systems
EMail: evoit@cisco.com
Requirements for Subscription to YANG Datastores
draft-ietf-i2rs-pub-sub-requirements-01

Abstract

This document provides requirements for a service that allows client applications to subscribe to updates of a YANG datastore. Based on criteria negotiated as part of a subscription, updates will be pushed to targeted recipients. Such a capability eliminates the need for periodic polling of YANG datastores by applications and fills a functional gap in existing YANG transports (i.e. Netconf and Restconf). Such a service can be summarized as a "pub/sub" service for YANG datastore updates. Beyond a set of basic requirements for the service, various refinements are addressed. These refinements include: periodicity of object updates, filtering out of objects underneath a requested a subtree, and delivery QoS guarantees.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
1. Introduction

YANG has gained acceptance as the data definition language of choice for control and management related information. Applications that interact with YANG datastores are extending beyond traditional configuration of network elements. In many cases these applications are aimed at service-assurance, which involves monitoring of operational data and state. The existing YANG technology ecosystem is proving insufficient for those applications due to:

- a reliance on RPC-style interactions where data is configured or fetched on-demand by applications.

- change notifications which identify a node associated with the config change, without the actual data updates.
Put simply, periodic fetching of data is not an adequate solution for applications requiring frequent or prompt updates of remote object state. Trying to impose a polling based solution to this problem imposes load on networks, devices, and applications. Additionally, polling solutions are brittle in the face of communication glitches, and they have limitations in their ability to synchronize and calibrate retrieval intervals across a network.

I2RS WG documents have expressed a need for more robust YANG object subscriptions. Similar discussions are underway in NETMOD and NETCONF. With the support of standards bodies such as OMG (DDS), XMPP.org standard, generic Pub/Sub mechanisms to communicate data updates have been defined and proven themselves in a wide variety of deployments.

It is time to incorporate such generic object subscription mechanisms as part of Network Elements, and allow these mechanisms to be applied in the context of data that is conceptually contained in YANG datastores. With such mechanisms, both controller and local Network Element based applications can have access to a set of consistent network information driven via push from peer Network Elements which host authoritative information.

There are some valid IETF starting points and contexts for these mechanisms. For example Netconf Event Notifications [RFC5277] provides a useful tool for an end-to-end solution. However RFC5277 does not follow the Pub/Sub paradigm, does not allow the explicit deletion of subscriptions, and predates YANG. [RFC6470] defines configuration change notifications, but doesn’t provide the actual configuration change.

Because of this, the authors have put forward this requirements document as well as [datastore-push]. We are hoping these could provide a context upon which to create new solution.

2. Business Drivers

For decades, information delivery of current network state has been accomplished either by fetching from operations interfaces, or via dedicated, customized networking protocols. With the growth of SDN, imperative policy distribution, and YANG’s ascent as a dominant programmatic interface to network elements, this mixture of fetch plus custom networking protocols is no longer sufficient. What is needed is a push mechanism that is able to deliver objects and object changes as they happen.

These push distribution mechanisms will not replace existing networking protocols. Instead they will supplement these protocols,
providing different response time, peering, scale, and security characteristics.

At the same time, SNMP and MIBs are still widely deployed and the de-facto choice for many monitoring solutions. Those solutions do not require support for configuration transactions and the need to validate and maintain configuration consistency, hence there is less pressure to abandon SNMP and MIBs. Arguably the biggest shortcoming of SNMP for those applications concerns the need to rely on periodic polling, because it introduces additional load on the network and devices, is brittle in case polling cycles are missed, and is hard to synchronize and calibrate across a network, making data obtained from multiple devices less comparable. If applications need to apply those same interaction patterns for YANG datastores, similar issues can be expected. Migration to YANG datastores by applications that do not have to worry about transactional integrity becomes a lot more compelling if those issues are addressed.

2.1. Pub/Sub in I2RS

Various I2RS documents highlight the need to provide Pub/Sub capabilities between network elements. From [i2rs-arch], there are references throughout the document beginning in section 6.2. Some specific examples include:

- section 7.6 provides high level pub/sub (notification) guidance
- section 6.4.2 identifies "subscribing to an information stream of route changes receiving notifications about peers coming up or going down"
- section 6.3 notes that when local config preempts I2RS, external notification might be necessary

In addition [i2rs-usecase] has relevant requirements. A small subset includes:

- L-Data-REQ-12: The I2RS interface should support user subscriptions to data with the following parameters: push of data synchronously or asynchronously via registered subscriptions...
- L-DATA-REQ-07: The I2RS interface (protocol and IMs) should allow a subscribe to select portions of the data model.
- PI-REQ01: monitor the available routes installed in the RIB of each forwarding device, including near real time notification of route installation and removal.
o BGP-REQ10: I2RS client SHOULD be able instruct the I2RS agent(s) to notify the I2RS client when the BGP processes on an associated routing system observe a route change to a specific set of IP Prefixes and associated prefixes....The I2RS agent should be able to notify the client via publish or subscribe mechanism.

o IGP-REQ-07: The I2RS interface (protocol and IMs) should support a mechanism where the I2RS Clients can subscribe to the I2RS Agent’s notification of critical node IGP events.

o MPLS-LDP-REQ-03: The I2RS Agent notifications should allow an I2RS client to subscribe to a stream of state changes regarding the LDP sessions or LDP LSPs from the I2RS Agent.

o L-Data-REQ-01: I2rs must be able to collect large data set from the network with high frequency and resolution with minimal impact to the device’s CPU and memory.

And [i2rs-traceability] has Pub/Sub requirements listed in Section 7.4.3.

o I2RS Agents SHOULD support publishing I2RS trace log information to that feed as described in [i2rs-arch]. Subscribers would then receive a live stream of I2RS interactions in trace log format and could flexibly choose to do a number of things with the log messages.

There are additional individual drafts such as [i2rs-pubsub-security] documenting the Pub/Sub needs for: time delivery sensitivity, support for multiple transport protocols, secure/authorized communications, and support for a range specification of subscribed data delivery content. So the list above should not be considered exhaustive.

2.2. Pub/Sub variants on Network Elements

Looking at history, there are many examples of switching and routing protocols which have done explicit or implicit pub/sub in the past. In addition, new policy notification mechanisms which operate on Switches and Routers are being specified now. A very small subset of these includes:

o Routing Adjacencies in MPLS VPNs [RFC6513]

o OSPF Route Flooding [RFC2328]

o Multicast topology establishment protocols (IGMP, PIM, etc.)
o Audio-Video Bridging streams needing guaranteed latency
   [AVB-latency] (802.1Q-2011 Clause 35)

o Secure Automation and Continuous Monitoring (SACM)
   [sacm-requirements]

o "Peer Mount" subscriptions for configuration verification between peers[draft-voit-netmod]

Worthy of note in the list above is the wide variety of broadcast,
multicast, and unicast transports used. In addition some transports
are at L3, and some at L2. Therefore if we are going to attempt a
generic Pub/Sub mechanism, it will need to be structured so that it
may support alternative transports. Looking at the nearer term based
on current I2RS requirements, NETCONF should be our transport
starting point as it supports connection oriented/Unicast
communication. But we need to be prepared to decouple where viable
to support Multicast and Broadcast distribution as well.

2.3. Existing Generalized Pub/Sub Implementations

TIBCO, RSS, CORBA, and other technologies all show precursor Pub/Sub
technologies. However there are new needs described in Section 4
below which these technologies do not serve. We need a technology.

There are at least two widely deployed generalized pub/sub
implementations which come close to current needs: XMPP[XEP-0060] and
DDS[OMG-DDS]. Both serve as proof-points that a highly scalable
distributed datastore implementation connecting millions of edge
devices is possible.

Because of these proof points, we can be comfortable that the
underlying technologies can enable reusable generalized YANG object
distribution. Analysis will need to fully dimension the speed and
scale of such object distribution for various subtree sizes and
transport types.

3. Terminology

A Subscriber makes requests for set(s) of YANG object data. The
Subscriber is the owner of the Subscription.

A Publisher is responsible for distributing subscribed YANG object
data per the terms of a Subscription. In general, a Publisher is the
owner of the YANG datastore that is subjected to the Subscription.

A Receiver is the target where a Publisher pushes updates. In
general, the Receiver and Subscriber will be the same entity. A
Subscription Service provides Subscriptions to Subscribers of YANG data.

A Subscription Service interacts with the Publisher of the YANG data as needed to provide the data per the terms of the Subscription.

A Subscription Request for one or more YANG subtrees made by the Subscriber of a Publisher and targeted to a Receiver. A Subscription MAY include constraints which dictates how often or under what conditions YANG subtree updates might be sent.

A Subscription is a contract between a Subscription Service and a Subscriber that stipulates the data to be pushed and the associated terms.

A YANG datastore is a conceptual datastore that contains hierarchical data defined in YANG data models. It is what is referred in existing RFCs as "Netconf datastore". However, as the same datastore is no longer tied to Netconf as a specific transport, the term "YANG datastore" is deemed more appropriate.

An Update provides object changes which have occurred within subscribed YANG subtree(s). An Update MUST include the current status of (data) node instances which according to any filtering are reportably different from the previously provided state. An Update MAY include a bundled set of ordered/sequential changes for a given object which have been made since the last update.

A Filter contains evaluation criteria which are evaluated against YANG object(s) within a Subscription. There are two types of Filters: Subtree Filters which identify selected objects/nodes published under a target data node, and object Property Filters where an object should only be published if it has propert(ies) meeting specified Filter criteria. For "on-change" notifications, passing through the Filter requires that a subscribed object is now different that from the previous Push, AND at least one of the YANG objects being evaluated has changed since the last Update.

4. Requirements

Many of the requirements within this section have been morphed from OMG’s DDS and XMPP.org’s requirements specifications.

4.1. Assumptions for Subscriber Behavior

This document provides requirements for the Subscription Service. It does not define all the requirements for the Subscriber/Receiver.
However In order to frame the desired behavior of the Subscription Service, it is important to specify key input constraints.

4.2. Subscription Service Requirements

This document provides requirements for the Subscription Service. It does not define all the requirements for the Subscriber/Receiver. However In order to frame the desired behavior of the Subscription Service, it is important to specify key input constraints.

A Subscriber SHOULD avoid attempting to establish multiple Subscriptions pertaining to the same information, i.e. referring to the same datastore YANG subtrees.

A Subscriber MAY provide QoS criteria to the Subscription Service such that if the Subscription Service is unable to meet those criteria, the Subscription should not be established.

When a Subscriber needs to restart, it is acceptable for the Subscriber to have to resubscribe. There is no requirement for the life span of the Subscription to extend beyond the life span of the Subscriber.

A Subscriber MUST be able to infer when a Subscription Service is no longer active and when no more updates are being sent.

A Subscriber MAY check with a Subscription Service to validate the existence and monitored subtrees of a Subscription.

A Subscriber MUST be able to periodically lease and re-lease a Subscription from a Subscription Service.

4.2.1. General

A Subscription Service MUST support the ability to create, renew, timeout, and terminate a Subscription.

A Subscription Service MUST be able to support and independently track one or more Subscription Requests by the same Subscriber.

A Subscription Service MUST be able to support an add/change/delete of one or more YANG subtrees as part of the same Subscription Request.

A Subscription Service MUST support Subscriptions against operational datastores, configuration datastores, or both.
A Subscription Service MUST be able support a Subtree Filter so that subscribed updates under a target node might publish only operational data, only configuration data, or both.

A Subscription MAY include filters as defined within a Subscription Request, the Subscription Service MUST publish only data nodes that meet the filter criteria.

A Subscription Service MUST support the ability to subscribe to periodic updates. The subscription period MUST be configurable as part of the subscription request.

A Subscription Service SHOULD support the ability to subscribe to updates "on-change", i.e. whenever values of subscribed data objects change.

For "on-change" updates, the Subscription Service MUST support a dampening period that needs to pass before the first or subsequent "on-change" updates are sent. The dampening period SHOULD be configurable as part of the subscription request.

A Subscription Service MUST allow Subscriptions to be monitored. Specifically, a Subscription Service MUST at a minimum maintain information about which Subscriptions are being serviced, the terms of those subscriptions (e.g. what data is being subscribed, associated filters, update policy - on change, periodic), and the overall status of the Subscription - e.g. active or suspended.

A Subscription Service SHOULD be able to interpret Subscription Requests QoS Policy requests, and only establish a Subscription if it is possible to meet the QoS those QoS Policy requests.

A Subscription Service MUST support terminating of a Subscription when requested by the Subscriber.

A Subscription Service SHOULD support the ability to suspend and to resume a Subscription on request of a client.

A Subscription Service MAY at its discretion revoke or suspend an existing subscription. Reasons MAY include transitory resource limitation, credential expiry, failure to reconfirm a subscription, loss of connectivity with the Receiver, operator CLI, and/or others. When this occurs, the Subscription Service MUST notify the Subscriber and update subscription status.

A Subscription Service MAY offer the ability to modify a subscription filter. If such an ability is offered, the service MUST provide
subscribers with an indication at what point the modified subscription goes into effect.

4.2.2. Negotiation

A Subscription Service MUST be able to negotiate the following terms of a Subscription:

- The policy: i.e. whether updates are on-change or periodic
- The interval, for periodic publication policy
- The dampening period, for on-change update policy
- Any filters associated with a subtree subscription

A Subscription Service SHOULD be able to negotiate QoS criteria for a Subscription. Examples of QoS criteria MAY include reliability of the Subscription Service, reaction time between a monitored YANG subtree/object change and a corresponding notification push, and the Subscription Service’s ability to support certain levels of object liveliness.

In cases where a Subscription Request cannot be fulfilled, the Subscription Service MUST include in its decline a set of criteria that would have been acceptable when the Subscription Request was made. For example, if periodic updates were requested with too short update intervals for the specified data set, the minimum acceptable interval period SHOULD be included. If on-change updates were requested with a dampening period, the minimum acceptable dampening period SHOULD be included, or an indication whether only periodic updates are supported along with the minimum acceptable interval period for the data set being subscribed to.

4.2.3. Update Distribution

For "on-change" updates, the Subscription Service MUST only send deltas to the object data for which a change occurred. [Otherwise the subscriber will not know what has actually undergone change.] The updates for each object needs to include an indication whether it was removed, added, or changed.

When a Subscription Service is not able to send updates per its subscription contract, the Subscription MUST notify subscribers and put the subscription into a state of indicating the Subscription was suspended by the service. When able to resume service, subscribers need to be notified as well. If unable to resume service, the
Subscription Service MAY terminate the subscription and notify Subscribers accordingly.

When a Subscription with "on-change" updates is suspended and then resumed, the first update SHOULD include updates of any changes that occurred while the Subscription was suspended, with the current value. The Subscription Service MUST provide a clear indication when this capability is not supported (because in this case a client application may have to synchronize state separately).

A Subscription Service MAY, as an option, support a persistence/replay capability.

4.2.4. Transport

A Subscription Service SHOULD support different transports.

A Subscription Service SHOULD support different encodings of payload.

It MUST be possible for Receivers to associate the update with a specific Subscription.

In the case of connection-oriented transport, when a transport connection drops, the associated Subscription SHOULD be terminated. It is up the Subscriber to request a new Subscription.

4.2.5. Security Requirements

As part of the Subscription establishment, there MUST be mutual authentication between the Subscriber and the Subscription Service.

When there are multiple Subscribers, it SHOULD be possible to provide cryptographic authentication in such a way that no Subscriber can pose as the original Subscription Service.

Versioning MUST be supported.

A Subscription could be used to attempt to retrieve information that a client has not 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 client MUST be filtered accordingly, just like if the data were being retrieved on-demand. For Unicast transports, the Netconf Authorization Control Model applies.

Subscription requests, including requests to create, terminate, suspend, and resume Subscriptions MUST be properly authorized.
When the Subscriber and Receiver are different, the Receiver MUST be able to terminate any Subscription to it where objects are being delivered over a Unicast transport.

A Subscription Service SHOULD decline a Subscription Request if it would deplete its resources. It is preferable to decline a Subscription when originally requested, rather than having to terminate it prematurely later.

4.2.6. QoS Requirements

A Subscription Service SHOULD be able to negotiate the following QoS parameters with a Subscriber: Dampening, Reliability, Deadline, Bundling.

4.2.6.1. Liveliness

A Subscription Service MUST be able to respond to requests to verify the Liveliness of a subscription.

A Subscription Service MUST be able to report the currently monitored Nodes of a Subscription.

4.2.6.2. Dampening

A Subscription Service MUST be able to negotiate the minimum time separation since the previous update before transmitting a subsequent update for Subscription. (Note: this is intended to confine the visibility of volatility into something digestible by the receiver.)

4.2.6.3. Reliability

A Subscription Service MAY send Updates over Best Effort and Reliable transports.

4.2.6.4. Coherence

Every update to a subscribed object MUST be sent to the Receiver in sequential order.

4.2.6.5. Presentation

The Subscription Service SHOULD have the ability to bundle a set of discrete object notifications into a single publishable update for a Subscription. A bundle MAY include information on different Data Nodes and/or multiple updates about a single Data Node.
For any bundled updates, the Subscription Service MUST provide information for a Receiver to reconstruct the order and timing of updates.

4.2.6.6. Deadline

The Subscription Service MUST be able to push updates at a regular cadence that corresponds with Subscriber specified start and end timestamps. (Note: the regular cadence can drive one, a discrete quantity, or an unbounded set of periodic updates.)

4.2.6.7. Push Latency

It MUST be possible for an administrative entity to determine the Push latency between object change in a monitored subtree and the Subscription Service Push of the update transmission.

4.2.7. Filtering

If no filtering criteria are provided, or if filtering criteria are met, updates for a subscribed object MUST be pushed, subject to the QoS limits established for the subscription.

It MUST be possible for the Subscription Service to receive Filter(s) from a Subscriber and apply them to corresponding object(s) within a Subscription.

It MUST be possible to attach one or more Subtree and/or Property Filters to a subscription. Mandatory Property Filter types include:

- For character based object properties, filter values which are exactly equal to a provided string, not equal to the string, or containing a string.

- For numeric based object properties, filter values which are =, !=, <, <=, >, >= a provided number.

It SHOULD be possible for Property Filtering criteria to evaluate more than one property of a particular subscribed object as well as apply multiple filters against a single property.

It SHOULD be possible to establish query match criteria on additional objects to be used in conjunction with Property Filtering criteria on a subscribed object. (For example: if A has changed AND B=1, then Push A.) (Note: Query match capability MAY be done on objects within the datastore even if those objects are not included within the subscription. This of course assumes the subscriber has read access to those objects.)
4.2.8. Assurance and Monitoring

It MUST be possible to fetch the state of a single subscription from a Subscription Service.

It MUST be possible to fetch the state of all subscriptions of a particular Subscriber.

It MUST be possible to fetch a list and status of all Subscription Requests over a period of time. If there us a failure, some failure reasons might include:

- Improper security credentials provided to access the target node
- Target node referenced does not exist
- Subscription type requested is not available upon the target node
- Out of resources, or resources not available
- Incomplete negotiations with the Subscriber.

5. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Ambika Tripathy and Prabhakara Yellai as well as the helpfulness of related end-to-end system context from [i2rs-pubsub-security] from Nancy Cam Winget, Ken Beck, and David McGrew.

6. References

6.1. Normative References


6.2. Informative References

[AVB-latency]


[XEP-0060]

[datastore-push]

[draft-voit-netmod]

[i2rs-arch]

[i2rs-pubsub-security]

[i2rs-traceability]


Authors’ Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Alex Clemm
Cisco Systems
Email: alex@cisco.com

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

This document presents NETCONF Call Home and RESTCONF Call Home, which respectively enable a NETCONF/RESTCONF server to initiate a secure connection to a NETCONF/RESTCONF client.

Editorial Note (To be removed by RFC Editor)

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

Artwork in this document contains placeholder references for this draft. Please apply the following replacement:

- "XXXX" --> the assigned RFC value for this draft

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

- draft-ietf-netconf-restconf
- draft-ietf-netconf-server-model

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

- "PORT-X" --> the assigned port value for "netconf-ch-ssh"
- "PORT-Y" --> the assigned port value for "netconf-ch-tls"
- "PORT-Z" --> the assigned port value for "restconf-ch-tls"
The following two Appendix sections are to be removed prior to publication:

- Appendix A. Change Log
- 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 August 6, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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. Requirements Terminology ................................ 4
   1.3. Applicability Statement ................................ 4
   1.4. Update to RFC 4253 .................................... 5
2. The NETCONF or RESTCONF Server .............................. 5
   2.1. Protocol Operation .................................. 5
1. Introduction

This document presents NETCONF Call Home and RESTCONF Call Home, which respectively enable a NETCONF/RESTCONF server to initiate a secure connection to a NETCONF/RESTCONF client. The NETCONF protocol is described in [RFC6241] and the RESTCONF is described in [draft-ietf-netconf-restconf].

Both NETCONF Call Home and RESTCONF Call Home preserve the client/server roles of underlying transport, as when compared to standard NETCONF and RESTCONF connections. Specifically, regardless if call home is used or not, the SSH and TLS client/server roles are the same. The SSH protocol is defined in [RFC4253] and the TLS protocol is defined in [RFC5246].

Ensuring consistency in the SSH and TLS roles is both necessary and desirable. Ensuring consistency is necessary for the SSH protocol, as SSH channels and subsystems can only be opened on the SSH server, which thus must always be the NETCONF server, in order to support NETCONF over SSH [RFC6242]. Ensuring consistency is desirable, for both the SSH and TLS protocols, as it conveniently leverages infrastructure that may be deployed for host-key or certificate verification and user authentication.

1.1. Motivation

Call home is generally useful for both the initial deployment and ongoing management of networking elements. Here are some scenarios enabled by call home:
The network element may proactively call home after being powered on for the first time in order to register itself with its management system.

The network element may access the network in a way that dynamically assigns it an IP address and it doesn’t register its assigned IP address to a mapping service, thus complicating the ability for a management system to connect to it.

The network element may be deployed behind a firewall that implements network address translation (NAT) for all internal network IP addresses, thus complicating the ability for a management system to connect to it.

The network element may be deployed behind a firewall that doesn’t allow any management access to the internal network.

The network element may be configured in "stealth mode" and thus doesn’t have any open ports for the management system to connect to.

The operator may prefer to have network elements initiate management connections, believing it is easier to secure one open-port in the data center than to have an open port on each network element in the network.

As the NETCONF and RESTCONF protocols become increasingly popular for programatic management of networking elements, having call home support for these two protocols is particularly desirable.

1.2. Requirements 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].

1.3. Applicability Statement

The techniques described in this document are suitable for network management scenarios such as the ones described in Section 1.1. However, these techniques SHOULD only be used for NETCONF Call Home and RESTCONF Call Home, as described in this document.

The reason for this restriction is that different protocols have different security assumptions. The NETCONF and RESTCONF protocols require clients and servers to verify the identity of the other party before starting the NETCONF/RESTCONF protocol (section 2.2 of [RFC6241] and sections 2.4 and 2.5 of [draft-ietf-netconf-restconf]).
This contrasts with the base SSH and TLS protocols, which do not require programmatic verification of the other party (section 9.3.4 of [RFC4251], section 4 of [RFC4252], and section 7.3 of [RFC5246]). In such circumstances, allowing the SSH/TLS server to contact the SSH/TLS client would open new vulnerabilities. Any use of call home with SSH/TLS for purposes other than NETCONF or RESTCONF will need a thorough, contextual security analysis.

1.4. Update to RFC 4253

This document updates the SSH Transport Layer Protocol [RFC4253] only in removing the "The client initiates the connection" statement made in Section 4 (Connection Setup). This document assumes that the reference to "connection" refers to the underlying transport connection (e.g., TCP), which the NETCONF server would initiate in a call home connection using the SSH protocol, even though it will not take on the role of the SSH client. Security implications related to this change are discussed in Security Considerations (Section 4).

2. The NETCONF or RESTCONF Server

2.1. Protocol Operation

- The NETCONF/RESTCONF server initiates a TCP connection request (SYN) to the NETCONF/RESTCONF client. The server SHOULD default to connecting to one of the IANA-assigned ports defined in section Section 5, but MAY be configured to use a non-default port.

- The TCP connection request is accepted and a TCP connection is established.

- Using this TCP connection, the NETCONF/RESTCONF server MUST immediately start using either the SSH-server [RFC4253] or the TLS-server [RFC5246] protocol, depending on how it is configured. For example, assuming the use of the IANA-assigned ports, the SSH-server protocol is used for PORT-X and the TLS-server protocol is used for either port PORT-Y or PORT-Z.

- Once the SSH or TLS connection is established, the NETCONF/RESTCONF server MUST immediately start using either the NETCONF-server [RFC6241] or RESTCONF-server [draft-ietf-netconf-restconf] protocol, depending on how it is configured. Assuming the use of the IANA-assigned ports, the NETCONF-server protocol is used for PORT-X or PORT-Y and the RESTCONF-server protocol is used for PORT-Z.
2.2. Configuration Data Model

How to configure a NETCONF or RESTCONF server to initiate a call home connection is outside the scope of this document, as implementations can support this protocol using proprietary configuration data models. That said, a YANG [RFC6020] model for configuring both NETCONF Call Home and RESTCONF Call Home is provided in [draft-ietf-netconf-server-model].

3. The NETCONF or RESTCONF Client

3.1. Protocol Operation

- The NETCONF/RESTCONF client listens for TCP connections from NETCONF/RESTCONF servers. The client SHOULD default to listening for connections on the IANA-assigned ports defined in section Section 5, but MAY be configured to use a non-default port.

- The NETCONF/RESTCONF client accepts an incoming TCP connection request and a TCP connection is established.

- Using this TCP connection, the NETCONF/RESTCONF client MUST immediately start using either the SSH-client [RFC4253] or the TLS-client [RFC5246] protocol, depending on how it is configured. For example, assuming the use of the IANA-assigned ports, the SSH-client protocol is used for PORT-X and the TLS-client protocol is used for either port PORT-Y or PORT-Z.

- Once the SSH or TLS connection is established, the NETCONF/RESTCONF client MUST immediately start using either the NETCONF-client [RFC6241] or RESTCONF-client [draft-ietf-netconf-restconf] protocol, depending on how it is configured. Assuming the use of the IANA-assigned ports, the NETCONF-client protocol is used for PORT-X or PORT-Y and the RESTCONF-client protocol is used for PORT-Z.

- The NETCONF protocol’s binding to SSH and TLS is defined in [RFC6242] and [RFC5539] respectively. The RESTCONF protocol’s binding to TLS is is defined in [RFC7230].
3.2. Server Identification and Verification

Under normal circumstances, a NETCONF/RESTCONF client initiates the connection to the NETCONF/RESTCONF server. This action provides essential input used to verify the NETCONF/RESTCONF server’s identity. For instance, when using TLS, the input can be compared to the domain names and IP addresses encoded in X.509 certificates. Similarly, when using SSH, the input can be compared to information persisted previously.

However, when receiving a call home connection, the NETCONF/RESTCONF client does not have any context leading it to know the connection is from a particular NETCONF/RESTCONF server. Thus the NETCONF/RESTCONF client must derive the NETCONF/RESTCONF server’s identity using information provided by the network and the NETCONF/RESTCONF server itself. This section describes strategies a NETCONF/RESTCONF client can use to identify a NETCONF/RESTCONF server.

In addition to identifying a NETCONF/RESTCONF server, a NETCONF/RESTCONF client must also be able to verify the server’s identity. Verifying a NETCONF/RESTCONF server’s identity is necessary under normal circumstances but, due to call home being commonly used for newly deployed NETCONF/RESTCONF servers, how to verify its identity the very first time becomes a prominent concern. Therefore, this section also describes strategies a NETCONF/RESTCONF client can use to verify a NETCONF/RESTCONF server’s identity.

The first information a NETCONF/RESTCONF client learns from a call home connection is the IP address of the NETCONF/RESTCONF server, as provided by the source address of the TCP connection. This IP address could be used as an identifier directly, but doing so would only work in networks that use known static addresses, in which case a standard NETCONF/RESTCONF connection would have worked just as well. Due to this limited use, it is not recommended to identify a NETCONF/RESTCONF server based on its source IP address.

The next information a NETCONF/RESTCONF client learns is provided by the NETCONF/RESTCONF server in the form of a host-key or a certificate, for the SSH and TLS protocols respectively. Without examining the contents of the host-key or certificate, it is possible to form an identity for the NETCONF/RESTCONF server using it directly (e.g., a fingerprint). This works because each NETCONF/RESTCONF server is assumed to have a statistically unique public key, even in virtualized environments. This strategy also provides a mechanism to verify the identity of the NETCONF/RESTCONF server, in that a secure connection can only be established with the NETCONF/RESTCONF server having the matching private key. This strategy is commonly implemented by SSH clients, and could be used equally well by TLS-
based clients, such as may be required when the NETCONF/RESTCONF servers have self-signed certificates. This strategy is viable and useful when the NETCONF/RESTCONF servers call home using either SSH with standard RSA/DSA host-keys, or using TLS with self-signed certificates.

Yet another option for identifying a NETCONF/RESTCONF server is for its host key or certificate to encode its identity directly (e.g., within the "Subject" field). However, in order to trust the content encoded within a host-key or certificate, it must be signed by a certificate authority trusted by the NETCONF/RESTCONF client. This strategy’s use of PKI enables a NETCONF/RESTCONF client to transparently authenticate the NETCONF/RESTCONF server’s certificate, thus eliminating the need for manual authentication, as required by the previously discussed strategies. Elimination of manual steps is needed to achieve scalable solutions, however one can claim that this merely pushes equivalent work to provisioning the NETCONF/RESTCONF servers with signed certificates. This assessment is accurate in general, but not in the case where the manufacturer itself provisions the certificates, such as is described by [Std-802.1AR-2009]. When NETCONF/RESTCONF servers are pre-provisioned this way, NETCONF/RESTCONF clients can transparently authenticate NETCONF/RESTCONF servers using just the manufacturer’s trust anchor and a list of expected NETCONF/RESTCONF server identifiers, which could be provided along with shipping information. This strategy is recommended for all deployment scenarios.

In discussing the use of certificates, it is worth noting that TLS uses X.509 certificates by default. However, to use X.509 certificates with SSH, both the NETCONF client and server must support [RFC6187].

4. Security Considerations

The security considerations described throughout [RFC6242] and [RFC5539], and by extension [RFC4253], [RFC5246], and [draft-ietf-netconf-restconf] apply here as well.

This RFC deviates from standard SSH and TLS usage by having the SSH/TLS server initiate the underlying TCP connection. For SSH, [RFC4253] says "the client initiates the connection", whereas for TLS, [RFC5246] says it is layered on top of "some reliable transport protocol" without further attribution.

Not having the SSH/TLS client initiate the TCP connection means that it does not have a preconceived notion of the SSH/TLS server’s identity, and therefore must dynamically derive one from information...
provided by the network or the SSH/TLS server itself. Security
Considerations for strategies for this are described in Section 3.2.

An attacker could DoS the NETCONF/RESTCONF client by having it
perform computationally expensive operations, before deducing that
the attacker doesn't possess a valid key. This is no different than
any secured service and all common precautions apply (e.g.,
blacklisting the source address after a set number of unsuccessful
login attempts).

5. IANA Considerations

This document requests that IANA assigns three TCP port numbers in
the "Registered Port Numbers" range with the service names "netconf-
ch-ssh", "netconf-ch-tls", and "restconf-ch-tls". These ports will
be the default ports for NETCONF Call Home and RESTCONF Call Home
protocols. Below is the registration template following the rules in
[RFC6335].

Service Name: netconf-ch-ssh
Transport Protocol(s): TCP
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Description: NETCONF Call Home (SSH)
Reference: RFC XXXX
Port Number: PORT-X

Service Name: netconf-ch-tls
Transport Protocol(s): TCP
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Description: NETCONF Call Home (TLS)
Reference: RFC XXXX
Port Number: PORT-Y

Service Name: restconf-ch-tls
Transport Protocol(s): TCP
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Description: RESTCONF Call Home (TLS)
Reference: RFC XXXX
Port Number: PORT-Z

6. Acknowledgements

The author would like to thank for following for lively discussions
on list and in the halls (ordered by last name): Andy Bierman, Martin
Bjorklund, Mehmet Ersue, Wes Hardaker, Stephen Hanna, David

Watsen Expires August 6, 2015
7. References

7.1. Normative References


7.2. Informative References

[Std-802.1AR-2009]

[draft-ietf-netconf-server-model]
Appendix A. Change Log

A.1. 00 to 01
  o The term "TCP connection" is now used throughout.
  o The terms "network element" and "management system" are now only used in the Motivation section.
  o Restructured doc a little to create an Introduction section.
  o Fixed reference in Applicability Statement so it would work equally well for SSH and TLS.
  o Fixed reported odd wording and three references.

A.2. 01 to 02
  o Added call home support for the RESTCONF protocol.
  o Fixed paragraph 3 of Security Considerations to equally apply to the TLS protocol.

A.3. 02 to 03
  o Tried to improve readability (issue #6)
  o Removed "FIXME" in section 1.3 (issue #7)
  o Added RFC Editor notes (issue #8)
  o Removed "TCP session" term (issue #9)
  o Improved language for usage of IANA-assigned ports (issue #10)

A.4. 03 to 04
  o Replaced "verify credentials" with "verify identity" (issue #11)

Appendix B. Open Issues

All issues with this draft are tracked using GitHub issues. Please see: https://github.com/netconf-wg/call-home/issues to see currently opened issues.
Author’s Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net
RESTCONF Protocol
draft-ietf-netconf-restconf-04

Abstract

This document describes an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastores defined in NETCONF.

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 August 3, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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. Simple Subset of NETCONF Functionality ............... 5
   1.2. Data Model Driven API ................................ 6
   1.3. Terminology ............................................. 7
       1.3.1. NETCONF ........................................... 7
       1.3.2. HTTP .............................................. 8
       1.3.3. YANG ............................................. 9
       1.3.4. Terms ............................................ 9
       1.3.5. URI Template .................................... 11
       1.3.6. Tree Diagrams .................................. 11
2. Transport Protocol Requirements ............................... 11
   2.1. Integrity and Confidentiality ......................... 11
   2.2. HTTPS with X.509v3 Certificates ....................... 12
   2.3. Certificate Validation ................................ 12
   2.4. Authenticated Server Identity ........................ 12
   2.5. Authenticated Client Identity ......................... 12
3. Resources ..................................................... 13
   3.1. Root Resource Discovery ................................ 14
   3.2. RESTCONF Resource Types ............................... 15
   3.3. API Resource ........................................... 15
       3.3.1. (+restconf)/data ................................ 16
       3.3.2. (+restconf)/operations ......................... 17
   3.4. datastore Resource .................................... 17
       3.4.1. Edit Collision Detection ......................... 18
   3.5. Data Resource ........................................... 19
       3.5.1. Encoding Data Resource Identifiers in the Request URI 19
       3.5.2. Defaults Handling ............................... 22
   3.6. Operation Resource .................................... 22
       3.6.1. Encoding Operation Input Parameters ............ 23
       3.6.2. Encoding Operation Output Parameters ........... 24
   3.7. Schema Resource ....................................... 24
   3.8. Event Stream Resource ................................ 25
   3.9. Errors Media Type ..................................... 26
4. Operations ..................................................... 26
   4.1. OPTIONS ................................................. 27
   4.2. HEAD .................................................... 27
   4.3. GET ..................................................... 27
   4.4. POST .................................................... 28
       4.4.1. Create Resource Mode ......................... 29
       4.4.2. Invoke Operation Mode ......................... 30
   4.5. PUT ..................................................... 31
   4.6. PATCH ................................................... 32
       4.6.1. Plain Patch .................................... 32
1. Introduction

There is a need for standard mechanisms to allow Web applications to access the configuration data, operational data, data-model specific protocol operations, and notification events within a networking device, in a modular and extensible manner.

This document describes an HTTP [RFC7230] based protocol called RESTCONF, for accessing data defined in YANG [RFC6020], using datastores defined in NETCONF [RFC6241].

The NETCONF protocol defines configuration datastores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these datastores. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and notification events. RESTCONF uses HTTP operations to provide CRUD operations on a NETCONF datastore containing YANG-defined data. Since NETCONF protocol operations are not relevant, the user should not need any prior knowledge of NETCONF in order to use RESTCONF.

Configuration data and state data are exposed as resources that can be retrieved with the GET method. Resources representing configuration data can be modified with the DELETE, PATCH, POST, and PUT methods. Data is encoded with either XML [W3C.REC-xml-20081126] or JSON [RFC7158].

Data-model specific protocol operations defined with the YANG "rpc" statement can be invoked with the POST method. Data-model specific notification events defined with the YANG "notification" statement can be accessed.

1.1. Simple Subset of NETCONF Functionality

The framework and meta-model used for an HTTP-based API does not need to mirror those used by the NETCONF protocol, but it needs to be compatible with NETCONF. Instead, a simplified framework and protocol is needed that co-exists with the three NETCONF datastores (candidate, running, startup), but hides the complexity of multiple datastores from the client.

A simplified transaction model is needed that allows basic CRUD operations on a hierarchy of conceptual resources. This represents a limited subset of the transaction capabilities of the NETCONF protocol.

The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by YANG data models. These basic edit operations allow the running configuration to be altered in an all-or-none fashion. This is similar to the "rollback-on-error" capability in NETCONF. Edits are usually applied to one data resource instance at a time.

Applications that require more complex transaction capabilities might consider NETCONF instead of RESTCONF. The following transaction features are not directly provided in RESTCONF:

- datastore locking (full or partial)
- candidate datastore
- startup datastore
- validate operation
- confirmed-commit procedure
RESTCONF is not intended to replace NETCONF, but rather provide an additional simplified interface that follows REST principles and is compatible with a resource-oriented device abstraction.

The following figure shows the system components:

```
+-----------+           +-----------------+
|  Web app  | <-------> |                 |
+-----------+   HTTP    | network device  |
|                 |   +-----------+ |
|  NMS app    | <-------> |   datastore   |
+-----------+  NETCONF  |   +-----------+ |
|                 |                     |
+-----------------+                     |
```

1.2. Data Model Driven API

RESTCONF combines the simplicity of the HTTP protocol with the predictability and automation potential of a schema-driven API. Using YANG, a client can predict all resource endpoints, much like using URI Templates [RFC6570], but in a more holistic manner. This strategy obviates the need for responses provided by the server to contain HATEOAS links, originally described in Roy Fielding’s doctoral dissertation [rest-dissertation].

In contrast, a REST client using HATEOAS principles would not use any data modeling language to define the application-specific content of the API. The client would need to discover each new child resource as it traverses the URIs to discover the server capabilities. This approach has the following significant weaknesses with regards to control of complex networking devices:

- inefficient performance: configuration APIs will be quite complex and may require thousands of protocol messages to discover all the schema information. Typically the data type information has to be passed in the protocol messages, which is also wasteful overhead.

- no data model richness: without a data model, the schema-level semantics and validation constraints are not available to the application.

- no tool automation: API automation tools need some sort of content schema to function. Such tools can automate various programming and documentation tasks related to specific data models.

Data models such as YANG modules serve as an "API contract" that will be honored by the server. An application designer can code to the data model, knowing in advance important details about the exact
protocol operations and datastore content a conforming server implementation will support.

RESTCONF provides the YANG module capability information supported by the server, in case the client wants to use it. The URIs for custom protocol operations and datastore content are predictable, based on the YANG module definitions.

Operational experience with CLI and SNMP indicates that operators learn the 'location' of specific service or device related data and do not expect such information to be arbitrary and discovered each time the client opens a management session to a server.

The RESTCONF protocol operates on a conceptual datastore defined with the YANG data modeling language. The server lists each YANG module it supports using the "ietf-yang-library" YANG module, defined in [I-D.ietf-netconf-yang-library].

The conceptual datastore contents, data-model-specific operations and notification events are identified by this set of YANG modules. All RESTCONF content identified as either a data resource, operation resource, or event stream resource is defined with the YANG language.

The classification of data as configuration or non-configuration is derived from the YANG "config" statement. Data ordering behavior is derived from the YANG "ordered-by" statement.

The RESTCONF datastore editing model is simple and direct, similar to the behavior of the ":writable-running" capability in NETCONF. Each RESTCONF edit of a datastore resource is activated upon successful completion of the transaction.

1.3. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.3.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
1.3.2. HTTP

The following terms are defined in [RFC3986]:

- fragment
- path
- query

The following terms are defined in [RFC7230]:

- header
- message-body
- request-line
- request URI
- status-line

The following terms are defined in [RFC7231]:

- method
- request
- resource

The following terms are defined in [RFC7232]:
1.3.3. YANG

The following terms are defined in [RFC6020]:

- container
- data node
- key leaf
- leaf
- leaf-list
- list
- presence container (or P-container)
- RPC operation (now called protocol operation)
- non-presence container (or NP-container)
- ordered-by system
- ordered-by user

1.3.4. Terms

The following terms are used within this document:

- API resource: a resource with the media type "application/yang.api+xml" or "application/yang.api+json".
- data resource: a resource with the media type "application/yang.data+xml" or "application/yang.data+json". Containers, leafs, list entries and anyxml nodes can be data resources.
- datastore resource: a resource with the media type "application/yang.datastore+xml" or "application/yang.datastore+json". Represents a datastore.
- edit operation: a RESTCONF operation on a data resource using either a POST, PUT, PATCH, or DELETE method.
- event stream resource: This resource represents an SSE (Server-Sent Events) event stream. The content consists of text using the
media type "text/event-stream", as defined by the HTML5 specification. Each event represents one <notification> message generated by the server. It contains a conceptual system or data-model specific event that is delivered within a notification event stream. Also called a "stream resource".

- media-type: HTTP uses Internet media types [RFC2046] in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation.

- operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, headers, and message-body.

- operation resource: a resource with the media type "application/yang.operation+xml" or "application/yang.operation+json".

- patch: a generic PATCH request on the target datastore or data resource. The media type of the message-body content will identify the patch type in use.

- plain patch: a specific PATCH request type that can be used for simple merge operations.

- query parameter: a parameter (and its value if any), encoded within the query component of the request URI.

- RESTCONF capability: An optional RESTCONF protocol feature supported by the server, which is identified by an IANA registered NETCONF Capability URI, and advertised with an entry in the "capability" leaf-list in Section 9.3.

- retrieval request: a request using the GET or HEAD methods.

- target resource: the resource that is associated with a particular message, identified by the "path" component of the request URI.

- unified datastore: A conceptual representation of the device running configuration. The server will hide all NETCONF datastore details for edit operations, such as the "candidate" and "startup" capabilities.

- schema resource: a resource with the media type "application/yang". The YANG representation of the schema can be retrieved by the client with the GET method.

- stream list: the set of data resource instances that describe the event stream resources available from the server. This
information is defined in the "ietf-restconf-monitoring" module as the "stream" list. It can be retrieved using the target resource "+restconf/data/ietf-restconf-monitoring:restconf-state/streams/stream". The stream list contains information about each stream, such as the URL to retrieve the event stream data.

1.3.5. URI Template

Throughout this document, the URI template [RFC6570] syntax "+restconf" is used to refer to the RESTCONF API entry point outside of an example. See Section 3.1 for details.

For simplicity, all of the examples in this document assume "/restconf" as the discovered RESTCONF API root path.

1.3.6. Tree Diagrams

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

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

2. Transport Protocol Requirements

2.1. Integrity and Confidentiality

HTTP [RFC7230] is an application layer protocol that may be layered on any reliable transport-layer protocol. RESTCONF is defined on top of HTTP, but due to the sensitive nature of the information conveyed, RESTCONF requires that the transport-layer protocol provides both data integrity and confidentiality, such as are provided by the TLS protocol [RFC5246].
2.2. HTTPS with X.509v3 Certificates

Given the nearly ubiquitous support for HTTP over TLS [RFC7230], RESTCONF implementations MUST support the "https" URI scheme, which has the IANA assigned default port 443. Consistent with the exclusive use of X.509v3 certificates for NETCONF over TLS [draft-ietf-netconf-rfc5539bis-07], use of certificates in RESTCONF is also limited to X.509v3 certificates.

2.3. Certificate Validation

When presented an X.509 certificate, the RESTCONF peer MUST use X.509 certificate path validation [RFC5280] to verify the integrity of the certificate. The presented X.509 certificate MAY also be considered valid if it matches a locally configured certificate fingerprint. If X.509 certificate path validation fails and the presented X.509 certificate does not match a locally configured certificate fingerprint, the connection MUST be terminated as defined in [RFC5246].

2.4. Authenticated Server Identity

The RESTCONF client MUST carefully examine the certificate presented by the RESTCONF server to determine if it meets the client’s expectations. If the RESTCONF client has external information as to the expected identity of the RESTCONF server, the hostname check MAY be omitted. Otherwise, the RESTCONF client MUST check its understanding of the RESTCONF server hostname against the server’s identity as presented in the server certificate message. Matching is performed according to the rules and guidelines defined in [RFC6125]. If the match fails, the RESTCONF client MUST either ask for explicit user confirmation or terminate the connection with an indication that the RESTCONF server’s identity is suspect.

2.5. Authenticated Client Identity

The RESTCONF server MUST authenticate the client access to any protected resource using HTTP Authentication [RFC7235]. If the RESTCONF client is not authenticated to access a resource, the server MUST send a response with status code 401 (Unauthorized) and a WWW-Authenticate header field containing at least one challenge applicable to the target resource. The RESTCONF server MAY advertise support for any number of authentication schemes but, in order to ensure interoperability, the RESTCONF server MUST advertise at least one of the following authentication schemes:

- Basic [draft-ietf-httpauth-basicauth-update-03]
These authentication schemes are selected due to their similarity to authentication schemes supported by NETCONF. In particular, the Basic and Digest authentication schemes both directly provide an identity and verification of a shared secret, much like NETCONF over SSH, when using the SSH "password" authentication method [RFC4252]. Similarly, the ClientCertificate authentication scheme is much like NETCONF over TLS’s use of X.509 client-certificates. When using the ClientCertificate authentication scheme, the RESTCONF server MUST verify the identity of the RESTCONF client using the algorithm defined in section 7 of [draft-ietf-netconf-rfc5539bis-07].

The RESTCONF client identity determined from any HTTP authentication scheme is hereafter known as the "RESTCONF username" and subject to the NETCONF Access Control Module (NACM) [RFC6536].

3. Resources

The RESTCONF protocol operates on a hierarchy of resources, starting with the top-level API resource itself (Section 3.1). Each resource represents a manageable component within the device.

A resource can be considered a collection of conceptual data and the set of allowed methods on that data. It can contain nested child resources. The child resource types and methods allowed on them are data-model specific.

A resource has its own media type identifier, represented by the "Content-Type" header in the HTTP response message. A resource can contain zero or more nested resources. A resource can be created and deleted independently of its parent resource, as long as the parent resource exists.

All RESTCONF resources are defined in this document except specific datastore contents, protocol operations, and notification events. The syntax and semantics for these resource types are defined in YANG modules.

The RESTCONF resources are accessed via a set of URIs defined in this document. The set of YANG modules supported by the server will determine the data model specific operations, top-level data node resources, and notification event messages supported by the server.

The RESTCONF protocol does not include a resource discovery mechanism. Instead, the definitions within the YANG modules
advertised by the server are used to construct a predictable operation or data resource identifier.

3.1. Root Resource Discovery

In line with the best practices defined by [RFC7320], RESTCONF enables deployments to specify where the RESTCONF API is located. When first connecting to a RESTCONF server, a RESTCONF client MUST determine the root of the RESTCONF API. The client discovers this by getting the "/.well-known/host-meta" resource ([RFC6415]) and using the <Link> element containing the "restconf" attribute:

Request
-------
GET /.well-known/host-meta users HTTP/1.1
Host: example.com
Accept: application/xrd+xml

Response
---------
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn

<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
  <Link rel='restconf' href='/restconf'/>
</XRD>

Once discovering the RESTCONF API root, the client MUST prepend it to any subsequent request to a RESTCONF resource. For instance, using the "/restconf" path discovered above, the client can now determine the operations supported by the server; e.g. in this example a custom "play" operation is supported:
3.2. RESTCONF Resource Types

The RESTCONF protocol defines a set of application specific media types to identify each of the available resource types. The following resource types are defined in RESTCONF:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Media Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>application/yang.api+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.api+json</td>
</tr>
<tr>
<td>Datastore</td>
<td>application/yang.datastore+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.datastore+json</td>
</tr>
<tr>
<td>Data</td>
<td>application/yang.data+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.data+json</td>
</tr>
<tr>
<td>Errors</td>
<td>application/yang.errors+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.errors+json</td>
</tr>
<tr>
<td>Operation</td>
<td>application/yang.operation+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.operation+json</td>
</tr>
<tr>
<td>Schema</td>
<td>application/yang</td>
</tr>
</tbody>
</table>

3.3. API Resource

The API resource contains the state and access points for the RESTCONF features. It is the top-level resource located at `/restconf` and has the media type "application/yang.api+xml" or "application/yang.api+json".
YANG Tree Diagram for an API Resource:

```
+--rw restconf
   +--rw data
   +--rw operations
```

The "application/yang.api" restconf-media-type extension in the "ietf-restconf" module defined in Section 8 is used to specify the structure and syntax of the conceptual child resources within the API resource.

This resource has the following child resources:

```
+----------------+--------------------------------|
| Child Resource | Description                    |
+----------------+--------------------------------|
| data           | Contains all data resources    |
| operations     | Data-model specific operations |
+----------------+--------------------------------|
```

RESTCONF API Resource

3.3.1. {+restconf}/data

This mandatory resource represents the combined configuration and operational data resources that can be accessed by a client. It cannot be created or deleted by the client. The datastore resource type is defined in Section 3.4.

Example:

This example request by the client would retrieve only the non-configuration data nodes that exist within the "library" resource, using the "content" query parameter (see Section 4.8.3).

```
GET /restconf/data/example-jukebox:jukebox/library?content=nonconfig HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:
3.3.2.  {+restconf}/operations

This optional resource is a container that provides access to the
data-model specific protocol operations supported by the server. The
server MAY omit this resource if no data-model specific operations
are advertised.

Any data-model specific operations defined in the YANG modules
advertised by the server MAY be available as child nodes of this
resource.

Operation resources are defined in Section 3.6.

3.4.  Datastore Resource

The "{+restconf}/data" subtree represents the datastore resource
type, which is a collection of configuration and operational data
nodes.

A "unified datastore" interface is used to simplify resource editing
for the client. The RESTCONF unified datastore is a conceptual
interface to the native configuration datastores that are present on
the device.

The underlying NETCONF datastores (i.e., candidate, running, startup)
can be used to implement the unified datastore, but the server design
is not limited to the exact datastore procedures defined in NETCONF.

The "candidate" and "startup" datastores are not visible in the
RESTCONF protocol. Transaction management and configuration
persistence are handled by the server and not controlled by the
client.
A datastore resource can only be written directly with the PATCH method. Only the configuration data resources within the datastore resource can be edited directly with all methods.

Each RESTCONF edit of a datastore resource is saved to non-volatile storage in an implementation-specific matter by the server. There is no guarantee that configuration changes are saved immediately, or that the saved configuration is always a mirror of the NETCONF running datastore, if the server also supports NETCONF.

3.4.1. Edit Collision Detection

Two "edit collision detection" mechanisms are provided in RESTCONF, for datastore and data resources.

3.4.1.1. Timestamp

The last change time is maintained and the "Last-Modified" ([RFC7232], section 2.2) header is returned in the response for a retrieval request. The "If-Unmodified-Since" header can be used in edit operation requests to cause the server to reject the request if the resource has been modified since the specified timestamp.

The server MUST maintain a last-modified timestamp for the top-level (+restconf)/data resource and SHOULD maintain last-modified timestamps for descendant resources. For all resources, the server MUST return the "Last-Modified" header when the resource is retrieved with the GET or HEAD methods. If the server does not maintain a timestamp for a resource, it MUST return the timestamp of the resource’s ancestor, a process that may recurse up to the top-level (+restconf)/data resource. Only changes to configuration data resources within the datastore affect the timestamp.

3.4.1.2. Entity tag

A unique opaque string is maintained and the "ETag" ([RFC7232], section 2.3) header is returned in the response for a retrieval request. The "If-Match" header can be used in edit operation requests to cause the server to reject the request if the resource entity tag does not match the specified value.

The server MUST maintain an entity tag for the top-level (+restconf)/data resource and SHOULD maintain entity tags for descendant resources. For all resources, the server MUST return the "ETag" header when the resource is retrieved with the GET or HEAD methods. If the server does not maintain an entity tag for a resource, it MUST return the entity tag of the resource’s ancestor, a process that may recurse up to the top-level (+restconf)/data
resource. Only changes to configuration data resources within the datastore affect the entity tag.

3.5. Data Resource

A data resource represents a YANG data node that is a descendant node of a datastore resource. Each YANG-defined data node can be uniquely targeted by the request-line of an HTTP operation. Containers, leafs, list entries and anyxml nodes are data resources.

The representation maintained for each data resource is the YANG defined subtree for that node. HTTP operations on a data resource affect both the targeted data node and all its descendants, if any.

For configuration data resources, the server MAY maintain a last-modified timestamp for the resource, and return the "Last-Modified" header when it is retrieved with the GET or HEAD methods. If maintained, the resource timestamp MUST be set to the current time whenever the resource or any configuration resource within the resource is altered.

For configuration data resources, the server MAY maintain a resource entity tag for the resource, and return the "ETag" header when it is retrieved as the target resource with the GET or HEAD methods. If maintained, the resource entity tag MUST be updated whenever the resource or any configuration resource within the resource is altered.

A data resource can be retrieved with the GET method. Data resources are accessed via the "(+restconf)/data" entry point. This sub-tree is used to retrieve and edit data resources.

A configuration data resource can be altered by the client with some or all of the edit operations, depending on the target resource and the specific operation. Refer to Section 4 for more details on edit operations.

The resource definition version for a data resource is identified by the revision date of the YANG module containing the YANG definition for the data resource.

3.5.1. Encoding Data Resource Identifiers in the Request URI

In YANG, data nodes are named with an absolute XPath expression, defined in [XPath], starting from the document root to the target resource. In RESTCONF, URL encoded path expressions are used instead.
A predictable location for a data resource is important, since applications will code to the YANG data model module, which uses static naming and defines an absolute path location for all data nodes.

A RESTCONF data resource identifier is not an XPath expression. It is encoded from left to right, starting with the top-level data node, according to the "api-path" rule in Section 3.5.1.1. The node name of each ancestor of the target resource node is encoded in order, ending with the node name for the target resource.

If a data node in the path expression is a YANG list node, then the key values for the list (if any) MUST be encoded according to the following rules:

- The key leaf values for a data resource representing a YANG list MUST be encoded using one path segment [RFC3986].
- If there is only one key leaf value, the path segment is constructed by having the list name followed by an "=" followed by the single key leaf value.
- If there are multiple key leaf values, the value of each leaf identified in the "key" statement is encoded in the order specified in the YANG "key" statement, with a comma separating them.
- All the components in the "key" statement MUST be encoded. Partial instance identifiers are not supported.
- Quoted strings are supported in the key leaf values. Quoted strings MUST be used to express empty strings. (example: list=foo,"",baz).
- The "list-instance" ABNF rule defined in Section 3.5.1.1 represents the syntax of a list instance identifier.
- Resource URI values returned in Location headers for data resources MUST identify the module name, even if there are no conflicting local names when the resource is created. This ensures the correct resource will be identified even if the server loads a new module that the old client does not know about.

Examples:
container top {
    list list1 {
        key "key1 key2 key3";
        ...
        list list2 {
            key "key4 key5";
            ...
            leaf X { type string; }
        }
    }
}

For the above YANG definition, URI with key leaf values will be encoded as follows (line wrapped for display purposes only):

```
/restconf/data/example-top:top/list1=key1val,key2val,key3val3/
    list2=key4val,key5val/X
```

3.5.1.1. ABNF For Data Resource Identifiers

The "api-path" ABNF syntax is used to construct RESTCONF path identifiers:

```
api-path = "/" |
    "/" api-identifier
    0*("/" (api-identifier | list-instance )))

api-identifier = [module-name ":"] identifier  ;; note 1

module-name = identifier

list-instance = api-identifier ":=" key-value ["," key-value]*

key-value = string

string = <a quoted or unquoted or empty string>

;; An identifier MUST NOT start with
;; (('X'|'x') ('M'|'m') ('L'|'l'))
identifier = (ALPHA / "_")
    *(ALPHA / DIGIT / "_" / ":" / "," / ".")
```

Note 1: The syntax for "api-identifier" MUST conform to the JSON identifier encoding rules in section 4 of [I-D.ietf-netmod-yang-json].
3.5.2. Defaults Handling

RESTCONF requires that a server report its default handling mode (see Section 4.8.2 for details). If the optional "with-defaults" query parameter is supported by the server, a client may use it to control retrieval of default values (see Section 4.8.11 for details).

If the target of a GET method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the default value that is in use by the server.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MAY return the default values that are in use by the server, in accordance with its reported default handing mode and query parameters passed by the client.

3.6. Operation Resource

An operation resource represents a protocol operation defined with the YANG "rpc" statement.

All operation resources share the same module namespace as any top-level data resources, so the name of an operation resource cannot conflict with the name of a top-level data resource defined within the same module.

If 2 different YANG modules define the same "rpc" identifier, then the module name MUST be used in the request URI. For example, if "module-A" and "module-B" both defined a "reset" operation, then invoking the operation from "module-A" would be requested as follows:

```
POST /restconf/operations/module-A:reset HTTP/1.1
``` Server example.com

Any usage of an operation resource from the same module, with the same name, refers to the same "rpc" statement definition. This behavior can be used to design protocol operations that perform the same general function on different resource types.

If the "rpc" statement has an "input" section, then a message-body MAY be sent by the client in the request, otherwise the request message MUST NOT include a message-body. If the "rpc" statement has an "output" section, then a message-body MAY be sent by the server in the response, otherwise the response message MUST NOT include a
message-body in the response message, and MUST send a "204 No Content" status-line instead.

3.6.1. Encoding Operation Input Parameters

If the "rpc" statement has an "input" section, then the "input" node is provided in the message-body, corresponding to the YANG data definition statements within the "input" section.

Example:

The following YANG definition is used for the examples in this section.

```yang
rpc reboot {
    input {
        leaf delay {
            units seconds;
            type uint32;
            default 0;
        }
        leaf message { type string; }
        leaf language { type string; }
    }
}
```

The client might send the following POST request message:

```http
POST /restconf/operations/example-ops:reboot HTTP/1.1
Host: example.com
Content-Type: application/yang.operation+json

{
    "example-ops:input" : {
        "delay" : 600,
        "message" : "Going down for system maintenance",
        "language" : "en-US"
    }
}
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 25 Apr 2012 11:01:00 GMT
Server: example-server
```
3.6.2. Encoding Operation Output Parameters

If the "rpc" statement has an "output" section, then the "output" node is provided in the message-body, corresponding to the YANG data definition statements within the "output" section.

Example:

The following YANG definition is used for the examples in this section.

```yang
rpc get-reboot-info {
   output {
      leaf reboot-time {
         units seconds;
         type uint32;
      }
      leaf message { type string; }
      leaf language { type string; }
   }
}
```

The client might send the following POST request message:

```plaintext
POST /restconf/operations/example-ops:get-reboot-info HTTP/1.1
Host: example.com
Accept: application/yang.operation+json
```

The server might respond:

```plaintext
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang.operation+json

{
   "example-ops:output" : {
      "reboot-time" : 30,
      "message" : "Going down for system maintenance",
      "language" : "en-US"
   }
}
```

3.7. Schema Resource

The server can optionally support retrieval of the YANG modules it supports. To retrieve a YANG module, a client first needs to get the URL for retrieving the schema.
The client might send the following GET request message:

GET /restconf/data/ietf-yang-library:modules/module=example-jukebox,2014-07-03/schema HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:

HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang.data+json

{  
  "ietf-yang-library:schema":  
  "https://example.com/mymodules/example-jukebox/2014-07-03"
}

Next the client needs to retrieve the actual YANG schema.

The client might send the following GET request message:

GET https://example.com/mymodules/example-jukebox/2014-07-03
HTTP/1.1
Host: example.com
Accept: application/yang

The server might respond:

module example-jukebox {
  namespace "http://example.com/ns/example-jukebox";
  prefix "jbox";
  // rest of YANG module content deleted...
}

3.8. Event Stream Resource

An "event stream" resource represents a source for system generated event notifications. Each stream is created and modified by the server only. A client can retrieve a stream resource or initiate a long-poll server sent event stream, using the procedure specified in Section 6.3.

A notification stream functions according to the NETCONF Notifications specification [RFC5277]. The available streams can be
retrieved from the stream list, which specifies the syntax and semantics of a stream resource.

3.9. Errors Media Type

An "errors" media type is a collection of error information that is sent as the message-body in a server response message, if an error occurs while processing a request message. It is not considered a resource type because no instances can be retrieved with a GET request.

The "ietf-restconf" YANG module contains the "application/yang.errors" restconf-media-type extension which specifies the syntax and semantics of an "errors" media type. RESTCONF error handling behavior is defined in Section 7.

4. Operations

The RESTCONF protocol uses HTTP methods to identify the CRUD operation requested for a particular resource.

The following table shows how the RESTCONF operations relate to NETCONF protocol operations:

```
+----------+-------------------------------------+
| RESTCONF | NETCONF                             |
+----------+-------------------------------------+
   OPTIONS | none                                |
   HEAD    | none                                |
   GET     | <get-config>, <get>                 |
   POST    | <edit-config> (operation="create")  |
   PUT     | <edit-config> (operation="replace")|
   PATCH   | <edit-config> (operation="merge")   |
   DELETE  | <edit-config> (operation="delete")  |
+----------+-------------------------------------+
```

Table 1: CRUD Methods in RESTCONF

The NETCONF "remove" operation attribute is not supported by the HTTP DELETE method. The resource must exist or the DELETE method will fail. The PATCH method is equivalent to a "merge" operation when using a plain patch (see Section 4.6.1), other media-types may provide more granular control.

Access control mechanisms may be used to limit what operations can be used. In particular, RESTCONF is compatible with the NETCONF Access Control Model (NACM) [RFC6536], as there is a specific mapping between RESTCONF and NETCONF operations, defined in Table 1. The
resource path needs to be converted internally by the server to the corresponding YANG instance-identifier. Using this information, the server can apply the NACM access control rules to RESTCONF messages.

The server MUST NOT allow any operation to any resources that the client is not authorized to access.

Implementation of all methods (except PATCH) are defined in [RFC7231]. This section defines the RESTCONF protocol usage for each HTTP method.

4.1. OPTIONS

The OPTIONS method is sent by the client to discover which methods are supported by the server for a specific resource (e.g., GET, POST, DELETE, etc.).

The server SHOULD implement this method, however the same information could be extracted from the YANG modules and the RESTCONF protocol specification.

If the PATCH method is supported, then the "Accept-Patch" header MUST be supported and returned in the response to the OPTIONS request, as defined in [RFC5789].

4.2. HEAD

The HEAD method is sent by the client to retrieve just the headers that would be returned for the comparable GET method, without the response message-body. It is supported for all resource types, except operation resources.

The request MUST contain a request URI that contains at least the entry point component. The same query parameters supported by the GET method are supported by the HEAD method.

The access control behavior is enforced as if the method was GET instead of HEAD. The server MUST respond the same as if the method was GET instead of HEAD, except that no response message-body is included.

4.3. GET

The GET method is sent by the client to retrieve data and meta-data for a resource. It is supported for all resource types, except operation resources. The request MUST contain a request URI that contains at least the entry point component.
The server MUST NOT return any data resources for which the user does not have read privileges. If the user is not authorized to read the target resource, an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client.

If the user is authorized to read some but not all of the target resource, the unauthorized content is omitted from the response message-body, and the authorized content is returned to the client.

Example:

The client might request the response headers for a JSON representation of the "library" resource:

    GET /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album  HTTP/1.1
    Host: example.com
    Accept: application/yang.data+json

The server might respond:

    HTTP/1.1 200 OK
    Date: Mon, 23 Apr 2012 17:02:40 GMT
    Server: example-server
    Content-Type: application/yang.data+json
    Cache-Control: no-cache
    Pragma: no-cache
    ETag: a74eefc993a2b
    Last-Modified: Mon, 23 Apr 2012 11:02:14 GMT

    
    
    "example-jukebox:album" : [ 
    
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2011
    ] 

    

4.4. POST

The POST method is sent by the client to create a data resource or invoke an operation resource. The server uses the target resource media type to determine how to process the request.
### Resource Types that Support POST

#### 4.4.1. Create Resource Mode

If the target resource type is a datastore or data resource, then the POST is treated as a request to create a top-level resource or child resource, respectively. The message-body is expected to contain the content of a child resource to create within the parent (target resource). The data-model for the child tree is the subtree is defined by YANG for the child resource.

The "insert" and "point" query parameters are supported by the POST method for datastore and data resource types, as specified in the YANG definition in Section 8.

If the POST method succeeds, a "201 Created" status-line is returned and there is no response message-body. A "Location" header identifying the child resource that was created MUST be present in the response in this case.

If the user is not authorized to create the target resource, an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

To create a new "jukebox" resource, the client might send:

```
POST /restconf/data HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{ "example-jukebox:jukebox" : [null] }
```

If the resource is created, the server might respond as follows:
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Location: https://example.com/restconf/data/example-jukebox:jukebox
Last-Modified: Mon, 23 Apr 2012 17:01:00 GMT
ETag: b3a3e673be2

Refer to Appendix D.2.1 for more resource creation examples.

4.4.2. Invoke Operation Mode

If the target resource type is an operation resource, then the POST method is treated as a request to invoke that operation. The message-body (if any) is processed as the operation input parameters. Refer to Section 3.6 for details on operation resources.

If the POST request succeeds, a "200 OK" status-line is returned if there is a response message-body, and a "204 No Content" status-line is returned if there is no response message-body.

If the user is not authorized to invoke the target operation, an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

In this example, the client is invoking the "play" operation defined in the "example-jukebox" YANG module.

A client might send a "play" request as follows:

    POST /restconf/operations/example-jukebox:play HTTP/1.1
    Host: example.com
    Content-Type: application/yang.operation+json

    {  
        "example-jukebox:input": {  
            "playlist": "Foo-One",
            "song-number": 2
        }
    }

The server might respond:

    HTTP/1.1 204 No Content
    Date: Mon, 23 Apr 2012 17:50:00 GMT
    Server: example-server
4.5. PUT

The PUT method is sent by the client to create or replace the target resource.

The only target resource media type that supports PUT is the data resource. The message-body is expected to contain the content used to create or replace the target resource.

The "insert" (Section 4.8.6) and "point" (Section 4.8.7) query parameters are supported by the PUT method for data resources.

Consistent with [RFC7231], if the PUT request creates a new resource, a "201 Created" status-line is returned. If an existing resource is modified, either "200 OK" or "204 No Content" are returned.

If the user is not authorized to create or replace the target resource an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

An "album" child resource defined in the "example-jukebox" YANG module is replaced or created if it does not already exist.

To replace the "album" resource contents, the client might send as follows. Note that the request-line is wrapped for display purposes only:

```plaintext
PUT /restconf/data/example-jukebox:jukebox/
    library艺urist=Foo%20Fighters/album=Wasting%20Light   HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{  
  "example-jukebox:album" : {  
    "name" : "Wasting Light",  
    "genre" : "example-jukebox:alternative",  
    "year" : 2011  
  }  
}
```

If the resource is updated, the server might respond:
4.6. PATCH

RESTCONF uses the HTTP PATCH method defined in [RFC5789] to provide an extensible framework for resource patching mechanisms. It is optional to implement by the server. Each patch type needs a unique media type. Zero or more PATCH media types MAY be supported by the server. The media types supported by a server can be discovered by the client by sending an OPTIONS request (see Section 4.1).

If the target resource instance does not exist, the server MUST NOT create it.

If the PATCH request succeeds, a "200 OK" status-line is returned if there is a message-body, and "204 No Content" is returned if no response message-body is sent.

If the user is not authorized to alter the target resource an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

4.6.1. Plain Patch

The plain patch mechanism merges the contents of the message body with the target resource. If the target resource is a datastore resource (see Section 3.4), the message body MUST be either application/yang.datastore+xml or application/yang.datastore+json. If then the target resource is a data resource (see Section 3.5), then the message body MUST be either application/yang.data+xml or application/yang.data+json.

Plain patch can used to create or update, but not delete, a child resource within the target resource. Please see [I-D.ietf-netconf-yang-patch] for an alternate media-type supporting more granular control.

Example:

To replace just the "year" field in the "album" resource (instead of replacing the entire resource with the PUT method), the client might send a plain patch as follows. Note that the request-line is wrapped for display purposes only:

```
To replace just the "year" field in the "album" resource (instead of replacing the entire resource with the PUT method), the client might send a plain patch as follows. Note that the request-line is wrapped for display purposes only:
```
PATCH /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+json

{   "example-jukebox:album" : {   "year" : 2011
}
}

If the field is updated, the server might respond:

HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:49:30 GMT
ETag: b2788923da4c

The XML encoding for the same request might be:

PATCH /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+xml

<album xmlns="http://example.com/ns/example-jukebox">
    <year>2011</year>
</album>

4.7. DELETE

The DELETE method is used to delete the target resource. If the
DELETE request succeeds, a "204 No Content" status-line is returned,
and there is no response message-body.

If the user is not authorized to delete the target resource then an
error response containing a "403 Forbidden" or "404 Not Found"
status-line is returned to the client. All other error responses are
handled according to the procedures defined in Section 7.

Example:

To delete a resource such as the "album" resource, the client might
send:
DELETE /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com

If the resource is deleted, the server might respond:

HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:40 GMT
Server: example-server

4.8. Query Parameters

Each RESTCONF operation allows zero or more query parameters to be present in the request URI. The specific parameters that are allowed depends on the resource type, and sometimes the specific target resource used, in the request.

<table>
<thead>
<tr>
<th>Name</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>GET</td>
<td>Select config and/or non-config data resources</td>
</tr>
<tr>
<td>depth</td>
<td>GET</td>
<td>Request limited sub-tree depth in the reply content</td>
</tr>
<tr>
<td>fields</td>
<td>GET</td>
<td>Request a subset of the target resource contents</td>
</tr>
<tr>
<td>filter</td>
<td>GET</td>
<td>Boolean notification filter for event stream resources</td>
</tr>
<tr>
<td>insert</td>
<td>POST, PUT</td>
<td>Insertion mode for user-ordered data resources</td>
</tr>
<tr>
<td>point</td>
<td>POST, PUT</td>
<td>Insertion point for user-ordered data resources</td>
</tr>
<tr>
<td>start-time</td>
<td>GET</td>
<td>Replay buffer start time for event stream resources</td>
</tr>
<tr>
<td>stop-time</td>
<td>GET</td>
<td>Replay buffer stop time for event stream resources</td>
</tr>
<tr>
<td>with-defaults</td>
<td>GET</td>
<td>Control retrieval of default values</td>
</tr>
</tbody>
</table>

RESTCONF Query Parameters

Query parameters can be given in any order. Each parameter can appear at most once in a request URI. A default value may apply if the parameter is missing.

Refer to Appendix D.3 for examples of query parameter usage.
If vendors define additional query parameters, they SHOULD use a prefix (such as the enterprise or organization name) for query parameter names in order to avoid collisions with other parameters.

4.8.1. Query Parameter URIs

A new set of RESTCONF Capability URIs are defined to identify the specific query parameters and protocol features supported by the server.

<table>
<thead>
<tr>
<th>Name</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaults</td>
<td>urn:ietf:params:restconf:capability:defaults:1.0</td>
</tr>
<tr>
<td>depth</td>
<td>urn:ietf:params:restconf:capability:depth:1.0</td>
</tr>
<tr>
<td>fields</td>
<td>urn:ietf:params:restconf:capability:fields:1.0</td>
</tr>
<tr>
<td>filter</td>
<td>urn:ietf:params:restconf:capability:filter:1.0</td>
</tr>
<tr>
<td>insert</td>
<td>urn:ietf:params:restconf:capability:insert:1.0</td>
</tr>
<tr>
<td>replay</td>
<td>urn:ietf:params:restconf:capability:replay:1.0</td>
</tr>
<tr>
<td>with-</td>
<td>urn:ietf:params:restconf:capability:with-</td>
</tr>
<tr>
<td>defaults</td>
<td>defaults:1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESTCONF Query Parameter URIs

4.8.2. The "defaults" Protocol Capability URI

This URI identifies the defaults handling mode that is used by the server for processing default leafs in the unified datastore. A parameter named "basic-mode" is required for this capability URI. The "basic-mode" definitions are specified in the "With-Defaults Capability for NETCONF" [RFC6243].

This protocol capability URI MUST be supported by the server, and the MUST be listed in the "capability" leaf-list in Section 9.3.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report-all</td>
<td>No data nodes are considered default</td>
</tr>
<tr>
<td>trim</td>
<td>Values set to the YANG default-stmt value are</td>
</tr>
<tr>
<td></td>
<td>default</td>
</tr>
<tr>
<td>explicit</td>
<td>Values set by the client are never considered</td>
</tr>
<tr>
<td></td>
<td>default</td>
</tr>
</tbody>
</table>
If the "basic-mode" is set to "report-all" then the server MUST adhere to the defaults handling behavior defined in section 2.1 of [RFC6243].

If the "basic-mode" is set to "trim" then the server MUST adhere to the defaults handling behavior defined in section 2.2 of [RFC6243].

If the "basic-mode" is set to "explicit" then the server MUST adhere to the defaults handling behavior defined in section 2.3 of [RFC6243].

Example: (split for display purposes only)

```
urn:ietf:params:restconf:capability:defaults:1.0?
  basic-mode=explicit
```

4.8.3. The "content" Query Parameter

The "content" parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Return only configuration descendant data nodes</td>
</tr>
<tr>
<td>nonconfig</td>
<td>Return only non-configuration descendant data nodes</td>
</tr>
<tr>
<td>all</td>
<td>Return all descendant data nodes</td>
</tr>
</tbody>
</table>

This parameter is only allowed for GET methods on datastore and data resources. A 400 Bad Request error is returned if used for other methods or resource types.

The default value is determined by the "config" statement value of the requested data nodes. If the "config" value is "false", then the default for the "content" parameter is "nonconfig". If "config" is "true" then the default for the "content" parameter is "config".

This query parameter MUST be supported by the server.

4.8.4. The "depth" Query Parameter

The "depth" parameter is used to specify the number of nest levels returned in a response for a GET method. The first nest-level consists of the requested data node itself. Any child nodes which
are contained within a parent node have a depth value that is 1
greater than its parent.

The value of the "depth" parameter is either an integer between 1 and
65535, or the string "unbounded". "unbounded" is the default.

This parameter is only allowed for GET methods on API, datastore, and
data resources. A 400 Bad Request error is returned if it used for
other methods or resource types.

By default, the server will include all sub-resources within a
retrieved resource, which have the same resource type as the
requested resource. Only one level of sub-resources with a different
media type than the target resource will be returned.

If this query parameter is supported by the server, then the "depth"
query parameter URI MUST be listed in the "capability" leaf-list in
Section 9.3.

4.8.5. The "fields" Query Parameter

The "fields" query parameter is used to optionally identify data
nodes within the target resource to be retrieved in a GET method.
The client can use this parameter to retrieve a subset of all nodes
in a resource.

A value of the "fields" query parameter matches the following rule:

```
fields-expr = path '(' fields-expr / '*' ')' /
path ';'; fields-expr /
path
path = api-identifier [ '/' path ]
```

"api-identifier" is defined in Section 3.5.1.1.

";" is used to select multiple nodes. For example, to retrieve only
the "genre" and "year" of an album, use: "fields=genre;year".

Parentheses are used to specify sub-selectors of a node. For
example, to retrieve only the "label" and "catalogue-number" of an
album, use: "fields=admin(label;catalogue-number)".

"/" is used in a path to retrieve a child node of a node. For
example, to retrieve only the "label" of an album, use:
"fields=admin/label".
This parameter is only allowed for GET methods on api, datastore, and data resources. A 400 Bad Request error is returned if used for other methods or resource types.

If this query parameter is supported by the server, then the "fields" query parameter URI MUST be listed in the "capability" leaf-list in Section 9.3.

4.8.6. The "insert" Query Parameter

The "insert" parameter is used to specify how a resource should be inserted within a user-ordered list.

The allowed values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Insert the new data as the new first entry.</td>
</tr>
<tr>
<td>last</td>
<td>Insert the new data as the new last entry.</td>
</tr>
<tr>
<td>before</td>
<td>Insert the new data before the insertion point, as specified by the value of the &quot;point&quot; parameter.</td>
</tr>
<tr>
<td>after</td>
<td>Insert the new data after the insertion point, as specified by the value of the &quot;point&quot; parameter.</td>
</tr>
</tbody>
</table>

The default value is "last".

This parameter is only supported for the POST and PUT methods. It is only supported if the target resource is a data resource, and that data represents a YANG list or leaf-list that is ordered by the user.

If the values "before" or "after" are used, then a "point" query parameter for the insertion parameter MUST also be present, or a 400 Bad Request error is returned.

If this query parameter is supported by the server, then the "insert" query parameter URI MUST be listed in the "capability" leaf-list in Section 9.3. The "point" query parameter MUST also be supported by the server.

4.8.7. The "point" Query Parameter

The "point" parameter is used to specify the insertion point for a data resource that is being created or moved within a user ordered list or leaf-list.
The value of the "point" parameter is of type
"data-resource-identifier", defined in the "ietf-restconf" YANG
module Section 8.

This parameter is only supported for the POST and PUT methods. It is
also only supported if the target resource is a data resource, and
that data represents a YANG list or leaf-list that is ordered by the
user.

If the "insert" query parameter is not present, or has a value other
than "before" or "after", then a 400 Bad Request error is returned.

This parameter contains the instance identifier of the resource to be
used as the insertion point for a POST or PUT method.

If the server includes the "insert" query parameter URI in the
"capability" leaf-list in Section 9.3, then the "point" query
parameter MUST be supported.

4.8.8. The "filter" Query Parameter

The "filter" parameter is used to indicate which subset of all
possible events are of interest. If not present, all events not
precluded by other parameters will be sent.

This parameter is only allowed for GET methods on a text/event-stream
data resource. A 400 Bad Request error is returned if used for other
methods or resource types.

The format of this parameter is an XPath 1.0 expression, and is
evaluated in the following context:

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

- The function library is the core function library defined in XPath
  1.0.

- The set of variable bindings is empty.

- The context node is the root node.

The filter is used as defined in [RFC5277], section 3.6. If the
boolean result of the expression is true when applied to the
conceptual "notification" document root, then the notification event
is delivered to the client.
If this query parameter is supported by the server, then the "filter" query parameter URI MUST be listed in the "capability" leaf-list in Section 9.3.

4.8.9. The "start-time" Query Parameter

The "start-time" parameter is used to trigger the notification replay feature and indicate that the replay should start at the time specified. If the stream does not support replay, per the "replay-support" attribute returned by stream list entry for the stream resource, then the server MUST return the HTTP error code 400 Bad Request.

The value of the "start-time" parameter is of type "date-and-time", defined in the "ietf-yang" YANG module [RFC6991].

This parameter is only allowed for GET methods on a text/event-stream data resource. A 400 Bad Request error is returned if used for other methods or resource types.

If this parameter is not present, then a replay subscription is not being requested. It is not valid to specify start times that are later than the current time. If the value specified is earlier than the log can support, the replay will begin with the earliest available notification.

If this query parameter is supported by the server, then the "replay" query parameter URI MUST be listed in the "capability" leaf-list in Section 9.3. The "stop-time" query parameter MUST also be supported by the server.

If the "replay-support" leaf is present in the "stream" entry (defined in Section 9.3) then the server MUST support the "start-time" and "stop-time" query parameters for that stream.

4.8.10. The "stop-time" Query Parameter

The "stop-time" parameter is used with the replay feature to indicate the newest notifications of interest. This parameter MUST be used with and have a value later than the "start-time" parameter.

The value of the "stop-time" parameter is of type "date-and-time", defined in the "ietf-yang" YANG module [RFC6991].

This parameter is only allowed for GET methods on a text/event-stream data resource. A 400 Bad Request error is returned if used for other methods or resource types.
If this parameter is not present, the notifications will continue until the subscription is terminated. Values in the future are valid.

If this query parameter is supported by the server, then the "replay" query parameter URI MUST be listed in the "capability" leaf-list in Section 9.3. The "start-time" query parameter MUST also be supported by the server.

If the "replay-support" leaf is present in the "stream" entry (defined in Section 9.3) then the server MUST support the "start-time" and "stop-time" query parameters for that stream.

### 4.8.11. The "with-defaults" Query Parameter

The "with-defaults" parameter is used to specify how information about default data nodes should be returned in response to GET requests on data resources.

If the server supports this capability, then it MUST implement the behavior in section 4.5.1 of [RFC6243], except applied to the RESTCONF GET operation, instead of the NETCONF operations.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report-all</td>
<td>All data nodes are reported</td>
</tr>
<tr>
<td>trim</td>
<td>Data nodes set to the YANG default are not reported</td>
</tr>
<tr>
<td>explicit</td>
<td>Data nodes set by the client are not reported</td>
</tr>
<tr>
<td>report-all-tagged</td>
<td>All data nodes are reported and defaults are tagged</td>
</tr>
</tbody>
</table>

If the "with-defaults" parameter is set to "report-all" then the server MUST adhere to the defaults reporting behavior defined in section 3.1 of [RFC6243].

If the "with-defaults" parameter is set to "trim" then the server MUST adhere to the defaults reporting behavior defined in section 3.2 of [RFC6243].

If the "with-defaults" parameter is set to "explicit" then the server MUST adhere to the defaults reporting behavior defined in section 3.3 of [RFC6243].
If the "with-defaults" parameter is set to "report-all-tagged" then the server MUST adhere to the defaults reporting behavior defined in section 3.4 of [RFC6243].

If the "with-defaults" parameter is not present then the server MUST adhere to the defaults reporting behavior defined in its "basic-mode" parameter for the "defaults" protocol capability URI, defined in Section 4.8.2.

If the server includes the "with-defaults" query parameter URI in the "capability" leaf-list in Section 9.3, then the "with-defaults" query parameter MUST be supported.

5. Messages

The RESTCONF protocol uses HTTP entities for messages. A single HTTP message corresponds to a single protocol method. Most messages can perform a single task on a single resource, such as retrieving a resource or editing a resource. The exception is the PATCH method, which allows multiple datastore edits within a single message.

5.1. Request URI Structure

Resources are represented with URIs following the structure for generic URIs in [RFC3986].

A RESTCONF operation is derived from the HTTP method and the request URI, using the following conceptual fields:

<OP> /<restconf>/<path>?<query>#<fragment>

^       ^        ^       ^         ^
|       |        |       |         |
method  entry  resource  query    fragment
M       M        O        O         I

M=mandatory, O=optional, I=ignored

<text> replaced by client with real values

- o method: the HTTP method identifying the RESTCONF operation requested by the client, to act upon the target resource specified in the request URI. RESTCONF operation details are described in Section 4.
o entry: the root of the RESTCONF API configured on this HTTP server, discovered by getting the ".well-known/host-meta" resource, as described in Section 3.1.

o resource: the path expression identifying the resource that is being accessed by the operation. If this field is not present, then the target resource is the API itself, represented by the media type "application/yang.api".

o query: the set of parameters associated with the RESTCONF message. These have the familiar form of "name=value" pairs. All query parameters are optional to implement by the server and optional to use by the client. Each query parameter is identified by a URI. The server MUST list the query parameter URIs it supports in the "capabilities" list defined in Section 9.3.

There is a specific set of parameters defined, although the server MAY choose to support query parameters not defined in this document. The contents of the any query parameter value MUST be encoded according to [RFC2396], section 3.4. Any reserved characters MUST be encoded with escape sequences, according to [RFC2396], section 2.4.

o fragment: This field is not used by the RESTCONF protocol.

When new resources are created by the client, a "Location" header is returned, which identifies the path of the newly created resource. The client MUST use this exact path identifier to access the resource once it has been created.

The "target" of an operation is a resource. The "path" field in the request URI represents the target resource for the operation.

5.2. Message Headers

There are several HTTP header lines utilized in RESTCONF messages. Messages are not limited to the HTTP headers listed in this section.

HTTP defines which header lines are required for particular circumstances. Refer to each operation definition section in Section 4 for examples on how particular headers are used.

There are some request headers that are used within RESTCONF, usually applied to data resources. The following tables summarize the headers most relevant in RESTCONF message requests:
### RESTCONF Request Headers

The following tables summarize the headers most relevant in RESTCONF message responses:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Response Content-Types that are acceptable</td>
</tr>
<tr>
<td>Content-Type</td>
<td>The media type of the request body</td>
</tr>
<tr>
<td>Host</td>
<td>The host address of the server</td>
</tr>
<tr>
<td>If-Match</td>
<td>Only perform the action if the entity matches ETag</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>Only perform the action if modified since time</td>
</tr>
<tr>
<td>If-Unmodified-Since</td>
<td>Only perform the action if un-modified since time</td>
</tr>
</tbody>
</table>

### RESTCONF Response Headers

The following tables summarize the headers most relevant in RESTCONF message responses:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Valid actions when 405 error returned</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>The cache control parameters for the response</td>
</tr>
<tr>
<td>Content-Type</td>
<td>The media type of the response message-body</td>
</tr>
<tr>
<td>Date</td>
<td>The date and time the message was sent</td>
</tr>
<tr>
<td>ETag</td>
<td>An identifier for a specific version of a resource</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>The last modified date and time of a resource</td>
</tr>
<tr>
<td>Location</td>
<td>The resource identifier for a newly created resource</td>
</tr>
</tbody>
</table>

### 5.3. Message Encoding

RESTCONF messages are encoded in HTTP according to [RFC7230]. The "utf-8" character set is used for all messages. RESTCONF message content is sent in the HTTP message-body.

Content is encoded in either JSON or XML format. A server MUST support XML encoding and MAY support JSON encoding. XML encoding rules for data nodes are defined in [RFC6020]. The same encoding rules are used for all XML content. JSON encoding rules are defined in [I-D.ietf-netmod-yang-json]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.
Request input content encoding format is identified with the Content-Type header. This field MUST be present if a message-body is sent by the client.

Response output content encoding format is identified with the Accept header in the request, or if it is not specified, the request input encoding format is used. If there was no request input, then the default output encoding is XML. File extensions encoded in the request are not used to identify format encoding.

5.4. RESTCONF Meta-Data

The RESTCONF protocol needs to retrieve the same meta-data that is used in the NETCONF protocol. Information about default leafs, last-modified timestamps, etc. are commonly used to annotate representations of the datastore contents. This meta-data is not defined in the YANG schema because it applies to the datastore, and is common across all data nodes.

This information is encoded as attributes in XML. JSON encoding of meta-data is defined in [I-D.lhotka-netmod-yang-metadata].

5.5. Return Status

Each message represents some sort of resource access. An HTTP "status-line" header line is returned for each request. If a 4xx or 5xx range status code is returned in the status-line, then the error information will be returned in the response, according to the format defined in Section 7.1.

5.6. Message Caching

Since the datastore contents change at unpredictable times, responses from a RESTCONF server generally SHOULD NOT be cached.

The server SHOULD include a "Cache-Control" header in every response that specifies whether the response should be cached. A "Pragma" header specifying "no-cache" MAY also be sent in case the "Cache-Control" header is not supported.

Instead of using HTTP caching, the client SHOULD track the "ETag" and/or "Last-Modified" headers returned by the server for the datastore resource (or data resource if the server supports it). A retrieval request for a resource can include the "If-None-Match" and/or "If-Modified-Since" headers, which will cause the server to return a "304 Not Modified" status-line if the resource has not changed. The client MAY use the HEAD method to retrieve just the message
headers, which SHOULD include the "ETag" and "Last-Modified" headers, if this meta-data is maintained for the target resource.

6. Notifications


6.1. Server Support

A RESTCONF server is not required to support RESTCONF notifications. Clients may determine if a server supports RESTCONF notifications by using the HTTP operation OPTIONS, HEAD, or GET on the stream list. The server does not support RESTCONF notifications if an HTTP error code is returned (e.g., 404 Not Found).

6.2. Event Streams

A RESTCONF server that supports notifications will populate a stream resource for each notification delivery service access point. A RESTCONF client can retrieve the list of supported event streams from a RESTCONF server using the GET operation on the stream list.

The "restconf-state/streams" container definition in the "ietf-restconf-monitoring" module (defined in Section 9.3) is used to specify the structure and syntax of the conceptual child resources within the "streams" resource.

For example:

The client might send the following request:

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

The server might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
```
<streams
    xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
    <stream>
        <name>NETCONF</name>
        <description>default NETCONF event stream</description>
        <replay-support>true</replay-support>
        <replay-log-creation-time>2007-07-08T00:00:00Z</replay-log-creation-time>
        <encoding>
            <type>xml</type>
            <events>https://example.com/streams/NETCONF</events>
        </encoding>
        <encoding>
            <type>json</type>
            <events>https://example.com/streams/NETCONF-JSON</events>
        </encoding>
    </stream>
    <stream>
        <name>SNMP</name>
        <description>SNMP notifications</description>
        <replay-support>false</replay-support>
        <encoding>
            <type>xml</type>
            <events>https://example.com/streams/SNMP</events>
        </encoding>
    </stream>
    <stream>
        <name>syslog-critical</name>
        <description>Critical and higher severity</description>
        <replay-support>true</replay-support>
        <replay-log-creation-time>2007-07-01T00:00:00Z</replay-log-creation-time>
        <encoding>
            <type>xml</type>
            <events>https://example.com/streams/syslog-critical</events>
        </encoding>
    </stream>
</streams>
6.3. Subscribing to Receive Notifications

RESTCONF clients can determine the URL for the subscription resource (to receive notifications) by sending an HTTP GET request for the "events" leaf with the stream list entry. The value returned by the server can be used for the actual notification subscription.

The client will send an HTTP GET request for the URL returned by the server with the "Accept" type "text/event-stream".

The server will treat the connection as an event stream, using the Server Sent Events [W3C.CR-eventsource-20121211] transport strategy.

The server MAY support query parameters for a GET method on this resource. These parameters are specific to each notification stream.

For example:

The client might send the following request:

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

The server might send the following response:

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

The RESTCONF client can then use this URL value to start monitoring the event stream:

```
GET /streams/NETCONF HTTP/1.1
Host: example.com
Accept: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
```

A RESTCONF client MAY request the server compress the events using the HTTP header field "Accept-Encoding". For instance:
6.3.1. NETCONF Event Stream

The server SHOULD support the "NETCONF" notification stream defined in [RFC5277]. For this stream, RESTCONF notification subscription requests MAY specify parameters indicating the events it wishes to receive. These query parameters are optional to implement, and only available if the server supports them.

<table>
<thead>
<tr>
<th>Name</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start-time</td>
<td>4.8.9</td>
<td>replay event start time</td>
</tr>
<tr>
<td>stop-time</td>
<td>4.8.10</td>
<td>replay event stop time</td>
</tr>
<tr>
<td>filter</td>
<td>4.8.8</td>
<td>boolean content filter</td>
</tr>
</tbody>
</table>

NETCONF Stream Query Parameters

The semantics and syntax for these query parameters are defined in the sections listed above. The YANG encoding MUST be converted to URL-encoded string for use in the request URI.

Refer to Appendix D.3.6 for filter parameter examples.

6.4. Receiving Event Notifications

RESTCONF notifications are encoded according to the definition of the event stream. The NETCONF stream defined in [RFC5277] is encoded in XML format.

The structure of the event data is based on the "notification" element definition in section 4 of [RFC5277]. It MUST conform to the schema for the "notification" element in section 4 of [RFC5277], except the XML namespace for this element is defined as:

urn:ietf:params:xml:ns:yang:ietf-restconf

For JSON encoding purposes, the module name is "ietf-restconf".

An example SSE notification encoded using XML:
An example SSE notification encoded using JSON:

```json
{  "ietf-restconf:notification": {    "event-time": "2013-12-21T00:01:00Z",    "example-mod:event": {      "event-class": "fault",      "reporting-entity": { "card": "Ethernet0" },      "severity": "major"    }  }
}
```

Alternatively, since neither XML nor JSON are whitespace sensitive, the above messages can be encoded onto a single line. For example:

For example: ('\' line wrapping added for formatting only)

XML:

```xml
<notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf"
  xmlns="http://example.com/event/1.0">
  <event-class>fault</event-class>
  <reporting-entity>
    <card>Ethernet0</card>
  </reporting-entity>
  <severity>major</severity>
</event>
</notification>
```

JSON:

```json
{  "event-time": "2013-12-21T00:01:00Z",  "example-mod:event": {    "event-class": "fault",    "reporting-entity": { "card": "Ethernet0" },    "severity": "major"  }}
```

The SSE specifications supports the following additional fields: event, id and retry. A RESTCONF server MAY send the "retry" field and, if it does, RESTCONF clients SHOULD use it. A RESTCONF server SHOULD NOT send the "event" or "id" fields, as there are no...
meaningful values that could be used for them that would not be redundant to the contents of the notification itself. RESTCONF servers that do not send the "id" field also do not need to support the HTTP header "Last-Event-Id". RESTCONF servers that do send the "id" field MUST still support the "startTime" query parameter as the preferred means for a client to specify where to restart the event stream.

7. Error Reporting

HTTP status-lines are used to report success or failure for RESTCONF operations. The <rpc-error> element returned in NETCONF error responses contains some useful information. This error information is adapted for use in RESTCONF, and error information is returned for "4xx" class of status codes.

The following table summarizes the return status codes used specifically by RESTCONF operations:

<table>
<thead>
<tr>
<th>Status-Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 Continue</td>
<td>POST accepted, 201 should follow</td>
</tr>
<tr>
<td>200 OK</td>
<td>Success with response message-body</td>
</tr>
<tr>
<td>201 Created</td>
<td>POST to create a resource success</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>POST to create a resource accepted</td>
</tr>
<tr>
<td>204 No Content</td>
<td>Success without response message-body</td>
</tr>
<tr>
<td>304 Not Modified</td>
<td>Conditional operation not done</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>Invalid request message</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>Access to resource denied</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>Resource target or resource node not found</td>
</tr>
<tr>
<td>405 Method Not Allowed</td>
<td>Method not allowed for target resource</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>Resource or lock in use</td>
</tr>
<tr>
<td>412 Precondition Failed</td>
<td>Conditional method is false</td>
</tr>
<tr>
<td>413 Request Entity Too Large</td>
<td>too-big error</td>
</tr>
<tr>
<td>414 Request-URI Too Large</td>
<td>too-big error</td>
</tr>
<tr>
<td>415 Unsupported Media</td>
<td>non RESTCONF media type</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>operation-failed</td>
</tr>
<tr>
<td>501 Not Implemented</td>
<td>unknown-operation</td>
</tr>
<tr>
<td>503 Service Unavailable</td>
<td>Recoverable server error</td>
</tr>
</tbody>
</table>

HTTP Status Codes used in RESTCONF
Since an operation resource is defined with a YANG "rpc" statement, a mapping between the NETCONF <error-tag> value and the HTTP status code is needed. The specific error condition and response code to use are data-model specific and might be contained in the YANG "description" statement for the "rpc" statement.

<table>
<thead>
<tr>
<th>&lt;error-tag&gt;</th>
<th>status code</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-use</td>
<td>409</td>
</tr>
<tr>
<td>invalid-value</td>
<td>400</td>
</tr>
<tr>
<td>too-big</td>
<td>413</td>
</tr>
<tr>
<td>missing-attribute</td>
<td>400</td>
</tr>
<tr>
<td>bad-attribute</td>
<td>400</td>
</tr>
<tr>
<td>unknown-attribute</td>
<td>400</td>
</tr>
<tr>
<td>bad-element</td>
<td>400</td>
</tr>
<tr>
<td>unknown-element</td>
<td>400</td>
</tr>
<tr>
<td>unknown-namespace</td>
<td>400</td>
</tr>
<tr>
<td>access-denied</td>
<td>403</td>
</tr>
<tr>
<td>lock-denied</td>
<td>409</td>
</tr>
<tr>
<td>resource-denied</td>
<td>409</td>
</tr>
<tr>
<td>rollback-failed</td>
<td>500</td>
</tr>
<tr>
<td>data-exists</td>
<td>409</td>
</tr>
<tr>
<td>data-missing</td>
<td>409</td>
</tr>
<tr>
<td>operation-not-supported</td>
<td>501</td>
</tr>
<tr>
<td>operation-failed</td>
<td>500</td>
</tr>
<tr>
<td>partial-operation</td>
<td>500</td>
</tr>
<tr>
<td>malformed-message</td>
<td>400</td>
</tr>
</tbody>
</table>

Mapping from error-tag to status code

7.1. Error Response Message

When an error occurs for a request message on a data resource or an operation resource, and a "4xx" class of status codes (except for status code "403 Forbidden"), then the server SHOULD send a response message-body containing the information described by the "errors" container definition within the YANG module Section 8. The Content-Type of this response message MUST be application/yang.errors (see example below).

The client MAY specify the desired encoding for error messages by specifying the appropriate media-type in the Accept header. If no error media is specified, the server MUST assume that "application/yang.errors+xml" was specified. All of the examples in this document, except for the one below, assume the default XML encoding will be returned if there is an error.
YANG Tree Diagram for <errors> Data:

```
+--ro errors
    +--ro error
        +--ro error-type       enumeration
        +--ro error-tag        string
        +--ro error-app-tag?   string
        +--ro (error-node)?
            +--:(error-path)
                | +--ro error-path?   instance-identifier
            +--:(error-urlpath)
                +--ro error-urlpath?   data-resource-identifier
        +--ro error-message?   string
        +--ro error-info
```

The semantics and syntax for RESTCONF error messages are defined in the "application/yang.errors" restconf-media-type extension in Section 8.

Examples:

The following example shows an error returned for an "lock-denied" error that can occur if a NETCONF client has locked a datastore. The RESTCONF client is attempting to delete a data resource. Note that an Accept header is used to specify the desired encoding for the error message. This example’s use of the Accept header is especially notable since the DELETE method typically doesn’t return a message-body and hence Accept headers are typically not passed.

```
DELETE /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
Accept: application/yang.errors+json
```

The server might respond:
HTTP/1.1 409 Conflict
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": "lock-denied",
         "error-message": "Lock failed, lock already held"
      }
   }
}

The following example shows an error returned for a "data-exists" error on a data resource. The "jukebox" resource already exists so it cannot be created.

The client might send:

POST /restconf/data/example-jukebox:jukebox HTTP/1.1
Host: example.com

The server might respond:

HTTP/1.1 409 Conflict
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": "data-exists",
         "error-urlpath": "https://example.com/restconf/data/example-jukebox:jukebox",
         "error-message": "Data already exists, cannot create new resource"
      }
   }
}

8. RESTCONF module

The "ietf-restconf" module defines conceptual definitions within an extension and two groupings, which are not meant to be implemented as datastore contents by a server. E.g., the "restconf" container is not intended to be implemented as a top-level data node (under the "/restconf/data" entry point).

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-restconf@2015-01-30.yang"

module ietf-restconf {
    namespace "urn:ietf:params:xml:ns:yang:ietf-restconf";
    prefix "rc";

    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:   Andy Bierman
                    <mailto:andy@yumaworks.com>
        Editor:   Martin Bjorklund
                    <mailto:mbj@tail-f.com>
        Editor:   Kent Watsen
                    <mailto:kwatsen@juniper.net>";

    description
        "This module contains conceptual YANG specifications
        for basic RESTCONF media type definitions used in
        RESTCONF protocol messages.

        Note that the YANG definitions within this module do not
        represent configuration data of any kind.
        The 'restconf-media-type' YANG extension statement
        provides a normative syntax for XML and JSON
        message"
encoding purposes.

Copyright (c) 2015 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-restconf-04.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-01-30 {
    description
    "Initial revision.";
    reference
    "RFC XXXX: RESTCONF Protocol.";
}

extension restconf-media-type {
    argument media-type-id {
        yin-element true;
    }
    description
    "This extension is used to specify a YANG data structure which represents a conceptual RESTCONF media type template. Data definition statements within this extension specify the 'generic syntax' for the specific media type.

    YANG is mapped to specific encoding formats outside the scope of this extension statement. RFC 6020 defines XML encoding rules for all RESTCONF media types that use the '+xml' suffix. draft-ietf-netmod-yang-json defines JSON encoding rules for all RESTCONF media types that use the 'json' suffix."
The ‘media-type-id’ parameter value identifies the media type that is being defined. It contains the string associated with the generic media type, i.e., no suffix is specified.

This extension is ignored unless it appears as a top-level statement. It SHOULD contain data definition statements that result in exactly one container data node definition. This allows compliant translation to an XML instance document for each media type.

The module name and namespace value for the YANG module using the extension statement is assigned to instance document data conforming to the data definition statements within this extension.

The sub-statements of this extension MUST follow the ‘data-def-stmt’ rule in the YANG ABNF.

The XPath document root is the extension statement itself, such that the child nodes of the document root are represented by the data-def-stmt sub-statements within this extension. This conceptual document is the context for the following YANG statements:

- must-stmt
- when-stmt
- path-stmt
- min-elements-stmt
- max-elements-stmt
- mandatory-stmt
- unique-stmt
- ordered-by
- instance-identifier data type

The following data-def-stmt sub-statements have special meaning when used within a restconf-resource extension statement.

- The list-stmt is not required to have a key-stmt defined.
- The if-feature-stmt is ignored if present.
- The config-stmt is ignored if present.
- The available identity values for any ‘identityref’ leaf or leaf-list nodes is limited to the module containing this extension statement, and the modules imported into that module.
typedef data-resource-identifier {
  type string {
    length "1 .. max";
  }
  description
    "Contains a Data Resource Identifier formatted string to identify a specific data resource instance. The document root for all data resources is a datastore resource container. Each top-level YANG data nodes supported by the server will be represented as a child node of the document root. The canonical representation of a data resource identifier includes the full server specification that is returned in the Location header when a new data resource is created with the POST method. The abbreviated representation does not contain any server location identification. Instead the identifier will start with the '/' character to represent the datastore document root for the data resource instance. The server MUST accept either representation and SHOULD return the canonical representation in any response message.";
  reference
    "RFC XXXX: [sec. 5.3.1.1 ABNF For Data Resource Identifiers]";
}

grouping errors {
  description
    "A grouping that contains a YANG container representing the syntax and semantics of a YANG Patch errors report within a response message.";
  container errors {
    description
      "Represents an error report returned by the server if
a request results in an error.

list error {
    description
    "An entry containing information about one specific error that occurred while processing a RESTCONF request.";
    reference "RFC 6241, Section 4.3";

    leaf error-type {
        type enumeration {
            enum transport {
                description "The transport layer";
            }
            enum rpc {
                description "The rpc or notification layer";
            }
            enum protocol {
                description "The protocol operation layer";
            }
            enum application {
                description "The server application layer";
            }
        }
        mandatory true;
        description
        "The protocol layer where the error occurred.";
    }

    leaf error-tag {
        type string;
        mandatory true;
        description
        "The enumerated error tag.";
    }

    leaf error-app-tag {
        type string;
        description
        "The application-specific error tag.";
    }

    choice error-node {
        description
        "The server will return the location of the error node in a format that is appropriate for the protocol. If no specific node within the request message body caused the error then this choice will not be present.";
    }
}
leaf error-path {
  type instance-identifier;
  description
  "The YANG instance identifier associated with the error node. This leaf will only be present if the error node is not a data resource, e.g., the error node is an input parameter for an operation resource.";
}
leaf error-urlpath {
  type data-resource-identifier;
  description
  "The target data resource identifier associated with the error node. This leaf will only be present if the error node is associated with a data resource (either within the server or in the request message).";
}
leaf error-message {
  type string;
  description
  "A message describing the error."
}
anyxml error-info {
  description
  "Arbitrary XML that represents a container of additional information for the error report."
}
}
} // grouping errors

grouping restconf {
  description
  "Conceptual container representing the application/yang.api resource type."
}
container restconf {
  description
  "Conceptual container representing the application/yang.api resource type."
}
container data {
  description

"Container representing the application/yang.datastore resource type. Represents the conceptual root of all operational data and configuration data supported by the server. The child nodes of this container can be any data resource (application/yang.data), which are defined as top-level data nodes from the YANG modules advertised by the server in the ietf-restconf-monitoring module."

container operations {
    description
    "Container for all operation resources (application/yang.operation),

    Each resource is represented as an empty leaf with the name of the RPC operation from the YANG rpc statement.

    E.g.;

    POST /restconf/operations/show-log-errors

    leaf show-log-errors {
        type empty;
    }

    ";

} // container restconf
} // grouping restconf

9. RESTCONF Monitoring

The "ietf-restconf-monitoring" module provides information about the RESTCONF protocol capabilities and notification event streams available from the server. Implementation is mandatory for RESTCONF servers, if any protocol capabilities or notification event streams are supported.

YANG Tree Diagram for "ietf-restconf-monitoring" module:
9.1. restconf-state/capabilities

This mandatory container holds the RESTCONF protocol capability URIs supported by the server.

The server MUST maintain a last-modified timestamp for this container, and return the "Last-Modified" header when this data node is retrieved with the GET or HEAD methods.

The server SHOULD maintain an entity-tag for this container, and return the "ETag" header when this data node is retrieved with the GET or HEAD methods.

9.2. restconf-state/streams

This optional container provides access to the notification event streams supported by the server. The server MAY omit this container if no notification event streams are supported.

The server will populate this container with a stream list entry for each stream type it supports. Each stream contains a leaf called "events" which contains a URI that represents an event stream resource.

Stream resources are defined in Section 3.8. Notifications are defined in Section 6.

9.3. RESTCONF Monitoring Module

The "ietf-restconf-monitoring" module defines monitoring information for the RESTCONF protocol.

The "ietf-yang-types" and "ietf-inet-types" modules from [RFC6991] are used by this module for some type definitions.
module ietf-restconf-monitoring {
  prefix "rcmon";

  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>
    WG Chair: Mehmet Ersue
      <mailto:mehmet.ersue@nsn.com>
    WG Chair: Mahesh Jethanandani
      <mailto:mjethanandani@gmail.com>
    Editor:   Andy Bierman
      <mailto:andy@yumaworks.com>
    Editor:   Martin Bjorklund
      <mailto:mbj@tail-f.com>
    Editor:   Kent Watsen
      <mailto:kwatsen@juniper.net>"

  description "This module contains monitoring information for the
    RESTCONF protocol."

  Copyright (c) 2015 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD License
  set forth in Section 4.c of the IETF Trust’s Legal Provisions
  Relating to IETF Documents
  (http://trustee.ietf.org/license-info)."
This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.

// RFC Ed.: replace XXXX with actual RFC number and remove this note.
// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-restconf-04.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-01-30 {
    description
        "Initial revision.";
    reference
        "RFC XXXX: RESTCONF Protocol.";
}

container restconf-state {
    config false;
    description
        "Contains RESTCONF protocol monitoring information.";

container capabilities {
    description
        "Contains a list of protocol capability URIs";

    leaf-list capability {
        type inet:uri;
        description "A RESTCONF protocol capability URI.";
    }
}

container streams {
    description
        "Container representing the notification event streams supported by the server.";
    reference
        "RFC 5277, Section 3.4, <streams> element.";

    list stream {
        key name;
        description
            "Each entry describes an event stream supported by the server.";

        leaf name {
            type string;
        }
    }
}
description "The stream name";
reference "RFC 5277, Section 3.4, <name> element.";

leaf description {
    type string;
    description "Description of stream content";
    reference
        "RFC 5277, Section 3.4, <description> element.";
}

leaf replay-support {
    type boolean;
    description
        "Indicates if replay buffer supported for this stream.
         If 'true', then the server MUST support the 'start-time'
         and 'stop-time' query parameters for this stream.";
    reference
        "RFC 5277, Section 3.4, <replaySupport> element.";
}

leaf replay-log-creation-time {
    when "../replay-support" {
        description
            "Only present if notification replay is supported";
    }
    type yang:date-and-time;
    description
        "Indicates the time the replay log for this stream
         was created.";
    reference
        "RFC 5277, Section 3.4, <replayLogCreationTime>
         element.";
}

list encoding {
    key type;
    min-elements 1;
    description
        "The server will create an entry in this list for each
         encoding format that is supported for this stream.
         The media type 'application/yang.stream' is expected
         for all event streams. This list identifies the
         sub-types supported for this stream.";

    leaf type {
        type string;
        description

"This is the secondary encoding format within the 'text/event-stream' encoding used by all streams. The type 'xml' is supported for the media type 'application/yang.stream+xml'. The type 'json' is supported for the media type 'application/yang.stream+json'."

leaf events {
  type inet:uri;
  mandatory true;
  description "Contains a URL that represents the entry point for establishing notification delivery via server sent events."
}

<CODE ENDS>

10. YANG Module Library

The "ietf-yang-library" module defined in [I-D.ietf-netconf-yang-library] provides information about the YANG modules and submodules used by the RESTCONF server. Implementation is mandatory for RESTCONF servers. All YANG modules and submodules used by the server MUST be identified in the YANG module library.

10.1. modules

This mandatory container holds the identifiers for the YANG data model modules supported by the server.

The server MUST maintain a last-modified timestamp for this container, and return the "Last-Modified" header when this data node is retrieved with the GET or HEAD methods.

The server SHOULD maintain an entity-tag for this container, and return the "ETag" header when this data node is retrieved with the GET or HEAD methods.
10.1.1. modules/module

This mandatory list contains one entry for each YANG data model module supported by the server. There MUST be an instance of this list for every YANG module that is used by the server.

The contents of this list are defined in the "module" YANG list statement in [I-D.ietf-netconf-yang-library].

The server MAY maintain a last-modified timestamp for each instance of this list entry, and return the "Last-Modified" header when this data node is retrieved with the GET or HEAD methods. If not supported then the timestamp for the parent "modules" container MAY be used instead.

The server MAY maintain an entity-tag for each instance of this list entry, and return the "ETag" header when this data node is retrieved with the GET or HEAD methods. If not supported then the timestamp for the parent "modules" container MAY be used instead.

11. IANA Considerations

11.1. The "restconf" Relation Type

This specification registers the "restconf" relation type in the Link Relation Type Registry defined by [RFC5988]:

Relation Name: restconf

Description: Identifies the root of RESTCONF API as configured on this HTTP server. The "restconf" relation defines the root of the API defined in RFCXXXX. Subsequent revisions of RESTCONF will use alternate relation values to support protocol versioning.

Reference: RFC XXXX

11.2. YANG Module Registry

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

This document registers two YANG modules in the YANG Module Names registry [RFC6020].

name:         ietf-restconf
prefix:       rc
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX

name:         ietf-restconf-monitoring
prefix:       rcmon
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX

11.3.  application/yang Media Sub Types

The parent MIME media type for RESTCONF resources is application/yang, which is defined in [RFC6020]. This document defines the following sub-types for this media type.
- api
- data
- datastore
- errors
- operation
- stream

Type name: application
Subtype name: yang.xxx

Required parameters: TBD
Optional parameters: TBD
Encoding considerations: TBD
Security considerations: TBD
Interoperability considerations: TBD

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

11.4. RESTCONF Capability URNs

[Note to RFC Editor: The RESTCONF Protocol Capability Registry does not yet exist; Need to ask IANA to create it; remove this note for publication]

This document registers several capability identifiers in "RESTCONF Protocol Capability URNs" registry
12. Security Considerations

This section provides security considerations for the resources defined by the RESTCONF protocol. Security considerations for HTTPS are defined in [RFC2818]. Security considerations for the content manipulated by RESTCONF can be found in the documents defining data models.

This document does not specify an authentication scheme, but it does require that an authenticated NETCONF username be associated with each HTTP request. The authentication scheme MAY be implemented in the underlying transport layer (e.g., client certificates) or within the HTTP layer (e.g., Basic Auth, OAuth, etc.). RESTCONF does not itself define an authentication mechanism, authentication MUST occur in a lower layer. Implementors SHOULD provide a comprehensive authorization scheme with RESTCONF and ensure that the resulting NETCONF username is made available to the RESTCONF server.

Authorization of individual user access to operations and data MAY be configured via NETCONF Access Control Model (NACM) [RFC6536], as specified in Section 4. Other authorization models MAY be used, but are outside of the scope of this document.
Configuration information is by its very nature sensitive. Its transmission in the clear and without integrity checking leaves devices open to classic eavesdropping and false data injection attacks. Configuration information often contains passwords, user names, service descriptions, and topological information, all of which are sensitive. Because of this, this protocol SHOULD be implemented carefully with adequate attention to all manner of attack one might expect to experience with other management interfaces.

Different environments may well allow different rights prior to and then after authentication. When an operation is not properly authorized, the RESTCONF server MUST return HTTP error status code 401 Unauthorized. Note that authorization information can be exchanged in the form of configuration information, which is all the more reason to ensure the security of the connection.

13. Acknowledgements

The authors would like to thank the following people for their contributions to this document: Ladislav Lhotka, Juergen Schoenwaelder, Rex Fernando, Robert Wilton, and Jonathan Hansford.

14. References

14.1. Normative References

[I-D.ietf-netconf-yang-library]  

[I-D.ietf-netmod-yang-json]  
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-02 (work in progress), November 2014.

[I-D.lhotka-netmod-yang-metadata]  
Lhotka, L., "Defining and Using Metadata with YANG", draft-lhotka-netmod-yang-metadata-00 (work in progress), September 2014.

[RFC2046]  
Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119]  
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.


Internet-Draft                  RESTCONF                    January 2015

[draft-ietf-httpauth-digest-09]

[draft-ietf-netconf-rfc5539bis-07]

[draft-thomson-httpbis-cant-01]

[rest-dissertation]

14.2. Informative References

[I-D.ietf-netconf-yang-patch]

[XPath]

Appendix A. Change Log
-- RFC Ed.: remove this section before publication.

The RESTCONF issue tracker can be found here: https://github.com/netconf-wg/restconf/issues

A.1. 03 - 04
  o renamed 'select' to 'fields' (#1)
  o moved collection resource and page capability to draft-ietf-netconf-restconf-collection-00 (#3)
  o added mandatory "defaults" protocol capability URI (#4)
Internet-Draft                  RESTCONF                    January 2015

- added optional "with-defaults" query parameter URI (#4)
- clarified authentication procedure (#9)
- moved ietf-yang-library module to draft-ietf-netconf-yang-library-00 (#13)
- clarified that JSON encoding of module name in a URI MUST follow
  the netmod-yang-json encoding rules (#14)
- added restconf-media-type extension (#15)
- remove 'content' query parameter URI and made this parameter
  mandatory (#16)
- clarified datastore usage
- changed lock-denied error example
- added with-defaults query parameter example
- added term "RESTCONF Capability"
- changed NETCONF Capability URI registry usage to new RESTCONF
  Capability URI Registry usage

A.2.  02 - 03

- added collection resource
- added "page" query parameter capability
- added "limit" and "offset" query parameters, which are available
  if the "page" capability is supported
- added "stream list" term
- fixed bugs in some examples
- added "encoding" list within the "stream" list to allow different
  <events> URLs for XML and JSON encoding.
- made XML MUST implement and JSON MAY implement for servers
- re-add JSON notification examples (previously removed)
- updated JSON references
A.3. 01 - 02

- moved query parameter definitions from the YANG module back to the plain text sections
- made all query parameters optional to implement
- defined query parameter capability URI
- moved 'streams' to new YANG module (ietf-restconf-monitoring)
- added 'capabilities' container to new YANG module (ietf-restconf-monitoring)
- moved 'modules' container to new YANG module (ietf-yang-library)
- added new leaf 'module-set-id' (ietf-yang-library)
- added new leaf 'conformance' (ietf-yang-library)
- changed 'schema' leaf to type inet:uri that returns the location of the YANG schema (instead of returning the schema directly)
- changed 'events' leaf to type inet:uri that returns the location of the event stream resource (instead of returning events directly)
- changed examples for yang.api resource since the monitoring information is no longer in this resource
- closed issue #1 'select parameter' since no objections to the proposed syntax
- closed "encoding of list keys" issue since no objection to new encoding of list keys in a target resource URI.
- moved open issues list to the issue tracker on github

A.4. 00 - 01

- fixed content=nonconfig example (non-config was incorrect)
- closed open issue 'message-id'. There is no need for a message-id field, and RFC 2392 does not apply.
- closed open issue 'server support verification'. The headers used by RESTCONF are widely supported.
- removed encoding rules from section on RESTCONF Meta-Data. This is now defined in "I-D.lhotka-netmod-yang-json".

- added media type application/yang.errors to map to errors YANG grouping. Updated error examples to use new media type.

- closed open issue ‘additional datastores’. Support may be added in the future to identify new datastores.

- closed open issue ‘PATCH media type discovery’. The section on PATCH has an added sentence on the Accept-Patch header.

- closed open issue ‘YANG to resource mapping’. Current mapping of all data nodes to resources will be used in order to allow mandatory DELETE support. The PATCH operation is optional, as well as the YANG Patch media type.

- closed open issue ‘_self links for HATEOAS support’. It was decided that they are redundant because they can be derived from the YANG module for the specific data.

- added explanatory text for the ‘select’ parameter.

- added RESTCONF Path Resolution section for discovering the root of the RESTCONF API using the /.well-known/host-meta.

- added an "error" media type to for structured error messages

- added Secure Transport section requiring TLS

- added Security Considerations section

- removed all references to "REST-like"

A.5. bierman:restconf-04 to ietf:restconf-00

- updated open issues section

Appendix B. Open Issues

-- RFC Ed.: remove this section before publication.

The RESTCONF issues are tracked on github.com:

https://github.com/netconf-wg/restconf/issues
Appendix C. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface.

YANG Tree Diagram for "example-jukebox" Module

```yang
+--rw jukebox?
   +--rw library
      |  +--rw artist [name]
      |     +--rw name     string
      |     +--rw album [name]
      |        +--rw name     string
      |        +--rw genre?   identityref
      |        +--rw year?    uint16
      |        +--rw admin
      |           +--rw label?              string
      |           +--rw catalogue-number?   string
      |      +--rw song [name]
      |          +--rw name     string
      |          +--rw location string
      |          +--rw format?     string
      |          +--rw length?     uint32
      |      +--ro artist-count?   uint32
      |      +--ro album-count?    uint32
      |      +--ro song-count?     uint32
      +--rw playlist [name]
       +--rw name           string
       +--rw description?   string
       +--rw song [index]
       |     +--rw index    uint32
       |     +--rw id       instance-identifier
   +--rw player
      +--rw gap?   decimal64
```

rpcs:

```yang
+----x play
  +--ro input
     +--ro playlist       string
     +--ro song-number    uint32
```

C.1. example-jukebox YANG Module

```yml
module example-jukebox {

    namespace "http://example.com/ns/example-jukebox";
    prefix "jbox";
}
```

import ietf-restconf { prefix rc; }

organization "Example, Inc."
contact "support at example.com"
description "Example Jukebox Data Model Module"
revision "2014-07-03"

// abbreviated list of genre classifications
identity alternative {
  base genre;
  description "Alternative music";
}

identity blues {
  base genre;
  description "Blues music";
}

identity country {
  base genre;
  description "Country music";
}

identity jazz {
  base genre;
  description "Jazz music";
}

identity pop {
  base genre;
  description "Pop music";
}

identity rock {
  base genre;
  description "Rock music";
}

container jukebox {
  presence
    "An empty container indicates that the jukebox service is available";

description
  "Represents a jukebox resource, with a library, playlists, and a play operation.";
}
container library {

description "Represents the jukebox library resource.";

list artist {
  key name;
  
  description "Represents one artist resource within the
  jukebox library resource.";

  leaf name {
    type string {
      length "1 .. max";
    }
    
    description "The name of the artist.";
  }
}

list album {
  key name;
  
  description "Represents one album resource within one
  artist resource, within the jukebox library.";

  leaf name {
    type string {
      length "1 .. max";
    }
    
    description "The name of the album.";
  }

  leaf genre {
    type identityref { base genre; } 
    
    description "The genre identifying the type of music on
    the album.";
  }

  leaf year {
    type uint16 {
      range "1900 .. max";
    }
    
    description "The year the album was released";
  }
}

cartridge admin {
  
  description

"Administrative information for the album."

leaf label {
  type string;
  description "The label that released the album.";
}
leaf catalogue-number {
  type string;
  description "The album’s catalogue number.";
}

list song {
  key name;
  description "Represents one song resource within one
  album resource, within the jukebox library.";

  leaf name {
    type string {
      length "1 .. max";
    }
    description "The name of the song";
  }
  leaf location {
    type string;
    mandatory true;
    description "The file location string of the
    media file for the song";
  }
  leaf format {
    type string;
    description "An identifier string for the media type
    for the file associated with the
    'location' leaf for this entry.";
  }
  leaf length {
    type uint32;
    units "seconds";
    description "The duration of this song in seconds.";
  }
  // end list 'song'
} // end list 'album'
} // end list 'artist'
leaf artist-count {
  type uint32;
  units "songs";
  config false;
  description "Number of artists in the library";
}
leaf album-count {
  type uint32;
  units "albums";
  config false;
  description "Number of albums in the library";
}
leaf song-count {
  type uint32;
  units "songs";
  config false;
  description "Number of songs in the library";
}
} // end library

list playlist {
  key name;

description
  "Example configuration data resource";

leaf name {
  type string;
  description
    "The name of the playlist.";
}
leaf description {
  type string;
  description
    "A comment describing the playlist.";
}
list song {
  key index;
  ordered-by user;

description
  "Example nested configuration data resource";

leaf index {  // not really needed
  type uint32;
  description
    "An arbitrary integer index for this playlist song.";
}
leaf id {
  type rc:data-resource-identifier;
  mandatory true;
  description "Song identifier. Must identify an instance of
   /jukebox/library/artist/album/song/name.";
}

} container player {
  description "Represents the jukebox player resource.";

  leaf gap {
    type decimal64 {
      fraction-digits 1;
      range "0.0 .. 2.0";
    }
    units "tenths of seconds";
    description "Time gap between each song";
  }

} rpc play {
  description "Control function for the jukebox player";
  input {
    leaf playlist {
      type string;
      mandatory true;
      description "playlist name";
    }
    leaf song-number {
      type uint32;
      mandatory true;
      description "Song number in playlist to play";
    }
  }
}
Appendix D. RESTCONF Message Examples

The examples within this document use the normative YANG module defined in Section 8 and the non-normative example YANG module defined in Appendix C.1.

This section shows some typical RESTCONF message exchanges.

D.1. Resource Retrieval Examples

D.1.1. Retrieve the Top-level API Resource

The client may start by retrieving the top-level API resource, using the entry point URI "{+restconf}".

```
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang.api+json
```

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.api+json

{
  "ietf-restconf:restconf": {
    "data": [ null ],
    "operations": {
      "play": [ null ]
    }
  }
}
```

To request that the response content to be encoded in XML, the "Accept" header can be used, as in this example request:

```
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang.api+xml
```

The server will return the same response either way, which might be as follows:

```
{...
```

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.api+xml

<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
    <data/>
    <operations>
        <play xmlns="https://example.com/ns/example-jukebox"/>
    </operations>
</restconf>

D.1.2. Retrieve The Server Module Information

In this example the client is retrieving the modules information from the server in JSON format:

GET /restconf/data/ietf-yang-library:modules HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond as follows.

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.data+json

{
    "ietf-yang-library:modules": {
        "module": [
            {
                "name": "foo",
                "revision": "2012-01-02",
                "schema": "https://example.com/mymodules/foo/2012-01-02",
                "namespace": "http://example.com/ns/foo",
                "feature": ["feature1", "feature2"],
                "conformance": true
            },
            {
                "name": "foo-types",
                "revision": "2012-01-05",
                "schema": 
            }
        ]
    }
}
"https://example.com/mymodules/foo-types/2012-01-05",
"schema" : [null],
"namespace" : "http://example.com/ns/foo-types",
"conformance" : false
},
{
"name" : "bar",
"revision" : "2012-11-05",
"schema" : "https://example.com/mymodules/bar/2012-11-05",
"namespace" : "http://example.com/ns/bar",
"feature" : [ "bar-ext" ],
"conformance" : true,
"submodule" : [ 
  {
    "name" : "bar-submod1",
    "revision" : "2012-11-05",
    "schema" : 
      "https://example.com/mymodules/bar-submod1/2012-11-05"
  },
  {
    "name" : "bar-submod2",
    "revision" : "2012-11-05",
    "schema" : 
      "https://example.com/mymodules/bar-submod2/2012-11-05"
  }
]
}
]
}

D.1.3. Retrieve The Server Capability Information

In this example the client is retrieving the capability information from the server in JSON format, and the server supports all the RESTCONF query parameters, plus one vendor parameter:

GET /restconf/data/ietf-restconf-monitoring:restconf-state/capabilities HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond as follows.
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.data+json

{
  "ietf-restconf-monitoring:capabilities": {
    "capability": [
      "urn:ietf:params:restconf:capability:content:1.0",
      "urn:ietf:params:restconf:capability:depth:1.0",
      "urn:ietf:params:restconf:capability:fields:1.0",
      "urn:ietf:params:restconf:capability:filter:1.0",
      "urn:ietf:params:restconf:capability:insert:1.0",
      "urn:ietf:params:restconf:capability:point:1.0",
      "urn:ietf:params:restconf:capability:start-time:1.0",
      "urn:ietf:params:restconf:capability:stop-time:1.0",
      "http://example.com/capabilities/myparam"
    ]
  }
}

D.2. Edit Resource Examples

D.2.1. Create New Data Resources

To create a new "artist" resource within the "library" resource, the client might send the following request.

POST /restconf/data/example-jukebox:jukebox/library HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:artist" : {
    "name" : "Foo Fighters"
  }
}

If the resource is created, the server might respond as follows. Note that the "Location" header line is wrapped for display purposes only:
To create a new "album" resource for this artist within the "jukebox" resource, the client might send the following request. Note that the request URI header line is wrapped for display purposes only:

```plaintext
POST /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters  HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
    "example-jukebox:album" : {
        "name" : "Wasting Light",
        "genre" : "example-jukebox:alternative",
        "year" : 2012   # note this is the wrong date
    }
}
```

If the resource is created, the server might respond as follows. Note that the "Location" header line is wrapped for display purposes only:

```plaintext
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:03:00 GMT
Server: example-server
Location: https://example.com/restconf/data/
    example-jukebox:jukebox/library/artist=Foo%20Fighters/
    album=Wasting%20Light
Last-Modified: Mon, 23 Apr 2012 17:03:00 GMT
ETag: b3830f23a4c
```

D.2.2. Detect Resource Entity Tag Change

In this example, the server just supports the mandatory datastore last-changed timestamp. The client has previously retrieved the "Last-Modified" header and has some value cached to provide in the following request to patch an "album" list entry with key value "Wasting Light". Only the "year" field is being updated.
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist=Foo%20Fighters/album=Wasting%20Light/year
HTTP/1.1
Host: example.com
Accept: application/yang.data+json
If-Unmodified-Since: Mon, 23 Apr 2012 17:01:00 GMT
Content-Type: application/yang.data+json

{ "example-jukebox:year" : "2011" }

In this example the datastore resource has changed since the time specified in the "If-Unmodified-Since" header. The server might respond:

HTTP/1.1 412 Precondition Failed
Date: Mon, 23 Apr 2012 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:45:00 GMT
ETag: b34aed893a4c

D.3. Query Parameter Examples

D.3.1. "content" Parameter

The "content" parameter is used to select the type of data child resources (configuration and/or not configuration) that are returned by the server for a GET method request.

In this example, a simple YANG list that has configuration and non-configuration child resources.

container events
   list event {
      key name;
      leaf name { type string; }
      leaf description { type string; }
      leaf event-count {
         type uint32;
         config false;
      }
   }

Example 1: content=all

To retrieve all the child resources, the "content" parameter is set to "all". The client might send:
GET /restconf/data/example-events:events?content=all
HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{  
  "example-events:events": {
    "event": [
      {
        "name": "interface-up",
        "description": "Interface up notification count",
        "event-count": 42
      },
      {
        "name": "interface-down",
        "description": "Interface down notification count",
        "event-count": 4
      }
    ]
  }
}

Example 2: content=config

To retrieve only the configuration child resources, the "content" parameter is set to "config" or omitted since this is the default value. Note that the "ETag" and "Last-Modified" headers are only returned if the content parameter value is "config".

GET /restconf/data/example-events:events?content=config
HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
ETag: eeeada438af
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-events:events": {
    "event": [
      {
        "name": "interface-up",
        "description": "Interface up notification count"
      },
      {
        "name": "interface-down",
        "description": "Interface down notification count"
      }
    ]
  }
}

Example 3: content=nonconfig

To retrieve only the non-configuration child resources, the "content" parameter is set to "nonconfig". Note that configuration ancestors (if any) and list key leafs (if any) are also returned. The client might send:

GET /restconf/data/example-events:events?content=nonconfig
HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-events:events" : {
    "event" : [
      {
        "name" : "interface-up",
        "event-count" : 42
      },
      {
        "name" : "interface-down",
        "event-count" : 4
      }
    ]
  }
}

D.3.2. "depth" Parameter

The "depth" parameter is used to limit the number of levels of child resources that are returned by the server for a GET method request.

This example shows how different values of the "depth" parameter would affect the reply content for retrieval of the top-level "jukebox" data resource.

Example 1: depth=unbounded

To retrieve all the child resources, the "depth" parameter is not present or set to the default value "unbounded". Note that some strings are wrapped for display purposes only.

GET /restconf/data/example-jukebox:jukebox?depth=unbounded
HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
{ "example-jukebox:jukebox": {
    "library": [
        "artist": [
            "name": "Foo Fighters",
            "album": [
                "name": "Wasting Light",
                "genre": "example-jukebox:alternative",
                "year": 2011,
                "song": [
                    "name": "Wasting Light",
                    "location": "/media/foo/a7/wasting-light.mp3",
                    "format": "MP3",
                    "length": 286
                ],
                { "name": "Rope",
                  "location": "/media/foo/a7/rope.mp3",
                  "format": "MP3",
                  "length": 259
                }
            ]
        ]
    ],
    "playlist": ["name": "Foo-One",
                  "description": "example playlist 1",
                  "song": [
                    { "index": 1,
                    "id": "https://example.com/restconf/data/example-jukebox:jukebox/library/artist=Foo%20Fighters/album/Wasting%20Light(song=Rope"
                  ],
                  { "index": 2,

Example 2: depth=1

To determine if 1 or more resource instances exist for a given target resource, the value "1" is used.

GET /restconf/data/example-jukebox:jukebox?depth=1 HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{"example-jukebox:jukebox": [null]}

Example 3: depth=3

To limit the depth level to the target resource plus 2 child resource layers the value "3" is used.

GET /restconf/data/example-jukebox:jukebox?depth=3 HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond:
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-jukebox:jukebox": {
    "library": {
      "artist": [ null ]
    },
    "playlist": [
      {
        "name": "Foo-One",
        "description": "example playlist 1",
        "song": [ null ]
      }
    ],
    "player": {
      "gap": 0.5
    }
  }
}

D.3.3. "fields" Parameter

In this example the client is retrieving the API resource, but retrieving only the "name" and "revision" nodes from each module, in JSON format:

GET /restconf/data?fields=modules/module(name;revision) HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond as follows.
D.3.4. "insert" Parameter

In this example, a new first entry in the "Foo-One" playlist is being created.

Request from client:

POST /restconf/data/example-jukebox:jukebox/
playlist=Foo-One?insert=first HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:song" : {
    "index" : 1,
    "id" : "/example-jukebox:jukebox/library/
             artist=Foo%20Fighters/album/Wasting%20Light/song/Rope"
  }
}

Response from server:
D.3.5. "point" Parameter

In this example, the client is inserting a new "song" resource within an "album" resource after another song. The request URI is split for display purposes only.

Request from client:

```bash
POST /restconf/data/example-jukebox:jukebox/
   library/artist=Foo%20Fighters/album/Wasting%20Light?
   insert=after&point=%2Fexample-jukebox%3Ajukebox%2F
   library%2Fartist%2FFoo%20Fighters%2Falbum%2F
   Wasting%20Light%2Fsong%2FBridge%20Burning

HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
   "example-jukebox:song" : {
      "name" : "Rope",
      "location" : "/media/foo/a7/rope.mp3",
      "format" : "MP3",
      "length" : 259
   }
}
```

Response from server:

HTTP/1.1 204 No Content

D.3.6. "filter" Parameter

The following URIs show some examples of notification filter specifications (lines wrapped for display purposes only):

```bash
```

// filter = /event/event-class='fault'
GET /mystreams/NETCONF?filter=%2Fevent%2Fevent-class%3D'fault'

// filter = /event/severity<=4
GET /mystreams/NETCONF?filter=%2Fevent%2Fseverity%3C%3D4

// filter = /linkUp|/linkDown
GET /mystreams/SNMP?filter=%2FlinkUp%7C%2FlinkDown

// filter = /*/reporting-entity/card!=‘Ethernet0’
GET /mystreams/NETCONF?
    filter=%2F%2Freporting-entity%2Fcard%21%3D'Ethernet0'

// filter = /*/email-addr[contains(.,'company.com')]
GET /mystreams/critical-syslog?
    filter=%2F%2Femail-addr[contains(.,%2C'company.com')]

// Note: the module name is used as prefix.
// filter = (/example-mod:event1/name='joe' and
//           /example-mod:event1/status='online')
GET /mystreams/NETCONF?
    filter=(%2Fexample-mod%3Aevent1%2Fname%3D'joe' and
            %2Fexample-mod%3Aevent1%2Fstatus%3D'online')

D.3.7.  "start-time" Parameter

// start-time = 2014-10-25T10:02:00Z
GET /mystreams/NETCONF?start-time=2014-10-25T10%3A02%3A00Z

D.3.8.  "stop-time" Parameter

// stop-time = 2014-10-25T12:31:00Z
GET /mystreams/NETCONF?stop-time=2014-10-25T12%3A31%3A00Z

D.3.9.  "with-defaults" Parameter

Assume the same data model as defined in Appendix A.1 of [RFC6243].
Assume the same data set as defined in Appendix A.2 of [RFC6243]. If
the server defaults-uri basic-mode is "trim", the the following
request for interface "eth1" might be as follows:

Without query parameter:

GET /restconf/data/interfaces/interface=eth1 HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond as follows.
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.data+json

{
  "example:interface": [
    {
      "name": "eth1",
      "status": "up"
    }
  ]
}

Note that the "mtu" leaf is missing because it is set to the default "1500", and the server defaults handling basic-mode is "trim".

With query parameter:

GET /restconf/data/interfaces/interface=eth1
   ?with-defaults=report-all HTTP/1.1
Host: example.com
Accept: application/yang.data+json

The server might respond as follows.

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.data+json

{
  "example:interface": [
    {
      "name": "eth1",
      "mtu": 1500,
      "status": "up"
    }
  ]
}

Note that the server returns the "mtu" leaf because the "report-all" mode was requested with the "with-defaults" query parameter.
Authors’ Addresses

Andy Bierman
YumaWorks
Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems
Email: mbj@tail-f.com

Kent Watsen
Juniper Networks
Email: kwatsen@juniper.net
Abstract

This document describes a collection resource for the RESTCONF protocol to provide enhanced filtering features for the retrieval of data nodes with the GET method.

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 August 3, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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
Table of Contents

1. Introduction ................................................. 2
   1.1. Terminology ......................................... 3
       1.1.1. NETCONF ....................................... 3
       1.1.2. HTTP .......................................... 4
       1.1.3. YANG ........................................... 5
       1.1.4. RESTCONF ...................................... 5
       1.1.5. Terms .......................................... 5
       1.1.6. URI Template ................................... 6
       1.1.7. Tree Diagrams .................................. 6
   1.2. Collection Resource Type .............................. 6
   1.3. Collection Resource ................................. 6
   1.4. Query Parameters ..................................... 7
       1.4.1. Query Parameter URIs ........................... 7
       1.4.2. The "limit" Query Parameter ................... 8
       1.4.3. The "offset" Query Parameter .................. 8
       1.4.4. The "sort" Query Parameter .................... 8
       1.4.5. The "where" Query Parameter ................... 8
2. RESTCONF Collection module ................................. 9
3. IANA Considerations ...................................... 11
   3.1. YANG Module Registry ................................ 11
   3.2. application/yang Media Sub Types .................... 11
   3.3. NETCONF Capability URNs .............................. 12
4. Security Considerations .................................. 12
5. Acknowledgements ......................................... 13
6. Normative References ..................................... 13
Appendix A. Change Log ..................................... 14
   A.1. restconf-03 to restconf-collection-00 ................ 14
Appendix B. Open Issues ................................... 14
Appendix C. RESTCONF Collection Examples .................. 14
   C.1. "limit" Parameter .................................. 14
   C.2. "offset" Parameter .................................. 15
   C.3. "sort" Parameter .................................... 16
   C.4. "where" Parameter .................................. 16
   C.5. "TopN" Use Case .................................... 16
Authors’ Addresses ........................................ 16

1. Introduction

There is a need for standard mechanisms to control the filtering, sorting, and retrieval of data from RESTCONF devices. A server may contain many instances of a particular YANG list. Retrieval of the entire list at once can be extremely inefficient.
Pagination mechanisms are needed to allow a client to iterate through a large list, in a manner that is most efficient for the application.

This document describes a "collection" resource that can be used to control retrieval of data nodes from a RESTCONF server.

[FIXME: describe basic needs
- target resource picks the list
- 'fields' is a node-set XPath expression to pick the subtrees within the target resource to return
- 'where' is a boolean XPath expression to pick which list instances are selected for return
- 'sort' is ??? parameter to sort the selected list instances
- 'limit' is the max number of list instances returned
- 'offset' is the XPath position() of the list instance ??? pre or post access control ???
  ??? if post, then what if NACM changes while client retrieving
]

Collection resources represent search results through the server data. Data that the client is not authorized to receive according to the access control parameters configured in [RFC6536] MUST NOT be returned in RESTCONF response messages.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

[FIXME: remove terms that are not used]

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
- datastore
- configuration datastore
o protocol operation
o running configuration datastore
o server
o startup configuration datastore
o state data
o user

1.1.2. HTTP

The following terms are defined in [RFC3986]:

o fragment
o path
o query

The following terms are defined in [RFC7230]:

o header
o message-body
o Request-Line
o request URI

The following terms are defined in [RFC7231]:

o method
o request
o resource

The following terms are defined in [RFC7232]:

o entity tag
1.1.3. YANG

The following terms are defined in [RFC6020]:

- container
- data node
- key leaf
- leaf
- leaf-list
- list
- presence container (or P-container)
- RPC operation (now called protocol operation)
- non-presence container (or NP-container)
- ordered-by system
- ordered-by user

1.1.4. RESTCONF

The following terms are defined in [I-D.ietf-netconf-restconf]:

- data resource
- target resource
- retrieval request

1.1.5. Terms

The following terms are used within this document:

- collection resource: a resource with the media type "application/yang.collection+xml" or "application/yang.collection+json". Contains a set of data resources.
1.1.6. URI Template

Throughout this document, the URI template [RFC6570] syntax "+restconf" is used to refer to the RESTCONF API entry point outside of an example. See the root resource discovery section defined in [I-D.ietf-netconf-restconf] for details.

All of the examples in this document assume "/restconf" as the discovered RESTCONF API root path.

1.1.7. Tree Diagrams

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

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

1.2. Collection Resource Type

The following resource type are defined in this document:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Media Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>application/yang.collection</td>
</tr>
</tbody>
</table>

RESTCONF Media Types

1.3. Collection Resource

A collection resource contains a set of data resources. It is used to represent a all instances or a subset of all instances in a YANG list or leaf-list.
A collection resource can be retrieved with the GET method, optionally with the query parameters "limit" (Section 1.4.2) and "offset" (Section 1.4.3).

The "ietf-restconf-collection" YANG module contains the "application/yang.collection" restconf-media-type extension which specifies the syntax and semantics of a "collection" media type.

1.4. Query Parameters

Each RESTCONF operation allows zero or more query parameters to be present in the request URI. The following query parameters are defined for RESTCONF collection resources.

<table>
<thead>
<tr>
<th>Name</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>GET</td>
<td>Number of entries to return for collection resources</td>
</tr>
<tr>
<td>offset</td>
<td>GET</td>
<td>Starting point for collection resources</td>
</tr>
<tr>
<td>sort</td>
<td>GET</td>
<td>Sorting criteria for collection resources</td>
</tr>
<tr>
<td>where</td>
<td>GET</td>
<td>Boolean filter to select data instances for a collection resource</td>
</tr>
</tbody>
</table>

RESTCONF Query Parameters

Query parameters can be given in any order. Each parameter can appear at most once in a request URI. A default value may apply if the parameter is missing.

Refer to Appendix C for examples of query parameter usage.

If vendors define additional query parameters, they SHOULD use a prefix (such as the enterprise or organization name) for query parameter names in order to avoid collisions with other parameters.

1.4.1. Query Parameter URIs

A new set of RESTCONF Capability URNs are defined to identify the specific query parameters supported by the server.
1.4.2. The "limit" Query Parameter

The "limit" parameter is used to restrict the number of data resources to return in response to GET requests on collection resources.

The value of the "limit" parameter is either an integer greater than or equal to 1, or the string "unbounded". The string "unbounded" is the default value.

If the server includes the "page" query parameter URI in the "capability" leaf-list in the "ietf-restconf-monitoring" module defined in [I-D.ietf-netconf-restconf], then the "limit" query parameter MUST be supported.

1.4.3. The "offset" Query Parameter

The "offset" parameter is used to specify the first data resource to return in response to GET requests on collection resources. Resources instances are numbered with consecutive integers from 1 to the number of resource instances.

The value of the "offset" parameter is an integer greater than or equal to 1. The default value is 1.

If the server includes the "page" query parameter URI in the "capability" leaf-list in the "ietf-restconf-monitoring" module defined in [I-D.ietf-netconf-restconf], then the "offset" query parameter MUST be supported.

1.4.4. The "sort" Query Parameter

[FIXME]

1.4.5. The "where" Query Parameter

[FIXME]
2. RESTCONF Collection module

The "ietf-restconf-collection" module defines conceptual definitions within groupings, which are not meant to be implemented as datastore contents by a server.

The "ietf-restconf" module from [I-D.ietf-netconf-restconf] is used by this module for the 'restconf-media-type' extension definition.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-restconf-collection@2015-01-30.yang"

module ietf-restconf-collection {
    prefix "rcoll";

    import ietf-restconf {
        prefix rc;
        revision-date 2015-01-30;
    }

    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: Andy Bierman
            <mailto:andy@yumaworks.com>
          Editor: Martin Bjorklund
            <mailto:mbj@tail-f.com>
          Editor: Kent Watsen
            <mailto:kwatsen@juniper.net>";

    description
        "This module contains conceptual YANG specifications for the RESTCONF Collection resource type.";

Note that the YANG definitions within this module do not represent configuration data of any kind. The YANG grouping statements provide a normative syntax for XML and JSON message encoding purposes.

Copyright (c) 2015 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

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

// RFC Ed.: remove this note
// Note: extracted from
// draft-ietf-netconf-restconf-collection-00.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-01-30 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: RESTCONF Collection Resource.";
}

rc:restconf-media-type "application/yang.collection" {
  uses collection;
}

grouping collection {
  description
    "Conceptual container representing the application/yang.collection resource type.";

collection {
  description
    "Container representing the application/yang.collection resource type.";
}
3. IANA Considerations

3.1. YANG Module Registry

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

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

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

name: ietf-restconf-collection
prefix: rcoll
// RFC Ed.: replace XXXX with RFC number and remove this note
reference: RFC XXXX

3.2. application/yang Media Sub Types

The parent MIME media type for RESTCONF resources is application/yang, which is defined in [RFC6020]. This document defines the following sub-types for this media type.
- collection

Type name: application

Subtype name: yang.xxx

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

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

Published specification: RFC XXXX

3.3. NETCONF Capability URNs

This document registers two capability identifiers in "RESTCONF Protocol Capability URNs" registry

<table>
<thead>
<tr>
<th>Capability Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>:page</td>
</tr>
<tr>
<td>urn:ietf:params:restconf:capability:page:1.0</td>
</tr>
<tr>
<td>:page-xpath</td>
</tr>
<tr>
<td>urn:ietf:params:restconf:capability:page-xpath:1.0</td>
</tr>
</tbody>
</table>

4. Security Considerations

This section provides security considerations for the resources defined by the RESTCONF protocol. Security considerations for HTTPS are defined in [RFC2818]. Security considerations for the content manipulated by RESTCONF can be found in the documents defining data models.

All security considerations that apply to resources defined in [I-D.ietf-netconf-restconf] also apply to the collection resource.
5. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Rex Fernando

6. Normative References

[I-D.ietf-netconf-restconf]


Appendix A. Change Log

-- RFC Ed.: remove this section before publication.

The RESTCONF issue tracker can be found here: https://github.com/netconf-wg/restconf/issues

A.1. restconf-03 to restconf-collection-00

- Moved collection resource from RESTCONF to a new document

Appendix B. Open Issues

-- RFC Ed.: remove this section before publication.

The RESTCONF Collection issues are tracked on github.com:

https://github.com/netconf-wg/restconf/issues

Appendix C. RESTCONF Collection Examples

The examples within this document use the "example-jukebox" YANG module defined in [I-D.ietf-netconf-restconf].

C.1. "limit" Parameter

In this example, the client requests the first two "album" resources for a given artist:

Request from client:

GET /restconf/data/example-jukebox:jukebox/
   library/artist=Foo%20Fighters/album/?limit=2 HTTP/1.1
Host: example.com
Content-Type: application/yang.collection+xml

Response from server:

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.collection+xml
<collection xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <album xmlns="http://example.com/ns/example-jukebox">
    <name>Foo Fighters</name>
    <year>1995</year>
    ...
  </album>
  <album xmlns="http://example.com/ns/example-jukebox">
    <name>The Color and the Shape</name>
    <year>1997</year>
    ...
  </album>
</collection>

C.2. "offset" Parameter

In this example, the client requests the next two albums, i.e., two albums starting from two.

Request from client:

GET /restconf/data/example-jukebox:jukebox/library/artist=Foo\%20Fighters/album/?limit=2&offset=2 HTTP/1.1
Host: example.com
Content-Type: application/yang.collection+json

Response from server:
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.collection+json

{
  "collection": {
    "example-jukebox:album": [
      {
        "year": 1999,
        "name": "There is Nothing Left to Loose",
        ...
      },
      {
        "year": 2002,
        "name": "One by One",
        ...
      }
    ]
  }
}

C.3. "sort" Parameter

[FIXME]

C.4. "where" Parameter

[FIXME]

C.5. "TopN" Use Case

[FIXME: use-case using all parameters for topN for some list]

Authors’ Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com
Abstract

This draft defines a NETCONF server configuration data model and a RESTCONF server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listens on, whether call-home is supported, and associated parameters.

Editorial Note (To be removed by RFC Editor)

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

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

- draft-ietf-netconf-rfc5539bis
- draft-ietf-netconf-restconf
- draft-ietf-netconf-call-home

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

- "VVVV" --> the assigned RFC value for this draft
- "WWWW" --> the assigned RFC value for draft-ietf-netconf-rfc5539bis
- "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf
o "YYYY" --> the assigned RFC value for draft-ietf-netconf-call-home
o "ZZZZ" --> the assigned RFC value for draft-thomson-httpbis-cant

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

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

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

- "2015-02-02" --> the publication date of this draft

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 6, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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. Terminology ........................................ 4
   1.2. Tree Diagrams ....................................... 4
2. Objectives .............................................. 5
   2.1. Support all NETCONF and RESTCONF transports ......... 5
   2.2. Enable each transport to select which keys to use .... 5
   2.3. Support authenticating NETCONF/RESTCONF clients certificates ........................................... 5
   2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames .................... 6
   2.5. Support both Listening for connections and Call Home .. 6
   2.6. For Call Home connections ............................ 6
       2.6.1. Support more than one northbound application .... 6
       2.6.2. Support applications having more than one server .. 6
       2.6.3. Support a reconnection strategy .................. 6
       2.6.4. Support both persistent and periodic connections .. 7
       2.6.5. Reconnection strategy for periodic connections .... 7
       2.6.6. Keep-alives for persistent connections ............ 7
       2.6.7. Customizations for periodic connections .......... 7
3. The NETCONF Server Configuration Model .................. 8
   3.1. Overview ............................................ 8
       3.1.1. The "session-options" subtree ..................... 8
       3.1.2. The "listen" subtree ............................. 8
       3.1.3. The "call-home" subtree .......................... 9
       3.1.4. The "ssh" subtree ................................ 11
       3.1.5. The "tls" subtree ................................ 11
   3.2. YANG Module ....................................... 12
4. The RESTCONF Server Configuration Model ................ 25
   4.1. Overview ........................................... 25
       4.1.1. The "listen" subtree ............................. 25
       4.1.2. The "call-home" subtree .......................... 26
       4.1.3. The "client-cert-auth" subtree ................... 28
   4.2. YANG Module ....................................... 28
5. Implementation strategy for keep-alives .................. 39
   5.1. Keep-alives for SSH ............................... 39
   5.2. Keep-alives for TLS ............................... 40
6. Security Considerations .................................. 40
7. IANA Considerations ..................................... 41
8. Other Considerations ..................................... 41
9. Acknowledgements ........................................ 42
10. References ................................................ 42
  10.1. Normative References ................................. 42
  10.2. Informative References ............................... 43
Appendix A. Examples .......................................... 44
  A.1. NETCONF Configuration using SSH Transport .......... 44
  A.2. NETCONF Configuration using TLS Transport .......... 45
  A.3. RESTCONF Configuration using TLS Transport .......... 47
Appendix B. Change Log ......................................... 47
  B.1. 00 to 01 .............................................. 47
  B.2. 01 to 02 .............................................. 48
  B.3. 02 to 03 .............................................. 48
  B.4. 03 to 04 .............................................. 48
  B.5. 04 to 05 .............................................. 48
  B.6. 05 to 06 .............................................. 49
Appendix C. Open Issues ........................................ 49

1. Introduction

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

1.1. Terminology

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

1.2. Tree Diagrams

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

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

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Objectives

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

2.1. Support all NETCONF and RESTCONF transports

The YANG module should support all current NETCONF and RESTCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [draft-ietf-netconf-rfc5539bis], and RESTCONF over TLS [draft-ietf-netconf-restconf], and to be extensible to support future transports as necessary.

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

2.2. Enable each transport to select which keys to use

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

2.3. Support authenticating NETCONF/RESTCONF clients certificates

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

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

2.5. Support both Listening for connections and Call Home

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

2.6. For Call Home connections

The following objectives only pertain to call home connections.

2.6.1. Support more than one northbound application

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

2.6.2. Support applications having more than one server

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

2.6.3. Support a reconnection strategy

Assuming an application has more than one server, then it becomes necessary to configure how a device should reconnect to the application should it lose its connection to the application’s servers. Of primary interest is if the device should start with
first server defined in a user-ordered list of servers or with the last server it was connected to. Secondary settings might specify the frequency of attempts and number of attempts per server. Therefore, a reconnection strategy should be configurable.

2.6.4. Support both persistent and periodic connections

Applications may vary greatly on how frequently they need to interact with a device, how responsive interactions with devices need to be, and how many simultaneous connections they can support. Some applications may need a persistent connection to devices to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for devices to initiate connections, the type of connection desired should be configurable.

2.6.5. Reconnection strategy for periodic connections

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

2.6.6. Keep-aliases 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 application requirements, and therefore keep-alive settings should be configurable on a per-application basis.

2.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the device to know how often it should connect. This delay essentially determines how long the application might have to wait to send data to the device. This setting does not constrain how often the device must wait to send data to the application, as the device should immediately connect to the application whenever it has data to send to it.
A common communication pattern is that one data transmission is many
times closely followed by another. For instance, if the device needs
to send a notification message, there’s a high probability that it
will send another shortly thereafter. Likewise, the application may
have a sequence of pending messages to send. Thus, it should be
possible for a device to hold a connection open until some amount of
time of no data being transmitted as transpired.

3. The NETCONF Server Configuration Model

3.1. Overview

3.1.1. The "session-options" subtree

```
module: ietf-netconf-server
  +--rw netconf-server
     +--rw session-options
        +--rw hello-timeout?  uint32
        +--rw idle-timeout?    uint32
```

The above subtree illustrates how the ietf-netconf-server YANG module
enables configuration of NETCONF session options, independent of any
transport or connection strategy. Please see the YANG module
(Section 3.2) for a complete description of these configuration
knobs.

3.1.2. The "listen" subtree
module: ietf-netconf-server
  +--rw netconf-server
     +--rw listen {listen}?
     |   +--rw max-sessions?   uint16
     |   +--rw endpoint* [name]
     |       +--rw name           string
     |       +--:(transport)
     |       |   +--:(ssh) {ssh}?
     |       |     |   +--rw ssh
     |       |     |       +--rw address?     inet:ip-address
     |       |     |       +--rw port?        inet:port-number
     |       |     |       +--rw host-keys
     |       |     |       |   +--rw host-key*   string
     |       |   +--:(tls) {tls}?
     |       |     +--rw tls
     |       |        +--rw address?     inet:ip-address
     |       |        +--rw port?        inet:port-number
     |       |        +--rw certificates
     |       |        |   +--rw certificate*   string
     |       +--rw keep-alives
     |       +--rw interval-secs?   uint8
     |       +--rw count-max?       uint8

The above subtree illustrates how the ietf-netconf-server YANG module enables configuration for listening for remote connections, as described in [RFC6242]. Feature statements are used to limit both if listening is supported at all as well as for which transports. If listening for connections is supported, then the model enables configuring a list of listening endpoints, each configured with a user-specified name (the key field), the transport to use (i.e. SSH, TLS), and the IP address and port to listen on. The port field is optional, defaulting to the transport-specific port when not configured. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.3. The "call-home" subtree
The above subtree illustrates how the ietf-netconf-server YANG module enables configuration for call home, as described in [draft-ietf-netconf-call-home]. Feature statements are used to limit both if call-home is supported at all as well as for which transports, if it is. If call-home is supported, then the model supports configuring a list of applications to connect to. Each application is configured with a user-specified name (the key field), the transport to be used (i.e. SSH, TLS), and a list of remote
endpoints, each having a name, an IP address, and an optional port. Additionally, the configuration for each remote application indicates the connection-type (persistent vs. periodic) and associated parameters, as well as the reconnection strategy to use. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.4. The "ssh" subtree

module: ietf-netconf-server

++rw netconf-server
    ++rw ssh {ssh}?
        +++rw x509 {ssh-x509-certs}?
            +++rw trusted-ca-certs
            | +++rw trusted-ca-cert* binary
            +++rw trusted-client-certs
                +++rw trusted-client-cert* binary

The above subtree illustrates how the ietf-netconf-server YANG module enables some SSH configuration independent of if the NETCONF server is listening or calling home. Specifically, when RFC 6187 is supported, this data model provides an ability to configure how client-certificates are authenticated. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.5. The "tls" subtree

module: ietf-netconf-server

++rw netconf-server
    ++rw tls {tls}?
        ++rw client-auth
            ++rw trusted-ca-certs
            | ++rw trusted-ca-cert* binary
            ++rw trusted-client-certs
            | ++rw trusted-client-cert* binary
            ++rw cert-maps
                ++rw cert-to-name* [id]
                    ++rw id uint32
                    ++rw fingerprint x509c2n:tls-fingerprint
                    ++rw map-type identityref
                    ++rw name string

The above subtree illustrates how the ietf-netconf-server YANG module enables TLS configuration independent of if the NETCONF server is listening or calling home. Specifically, this data model provides 1) an ability to configure how client-certificates are authenticated and 2) how authenticated client-certificates are mapped to NETCONF user
names. Please see the YANG module (Section 3.2) for a complete
description of these configuration knobs.

3.2. YANG Module

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

<CODE BEGINS> file "ietf-netconf-server@2015-02-02.yang"

module ietf-netconf-server {

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

  import ietf-netconf-acm {
    prefix nacm;                     // RFC 6536
    revision-date 2012-02-22;
  }
  import ietf-inet-types {           // RFC 6991
    prefix inet;
    revision-date 2013-07-15;
  }
  import ietf-x509-cert-to-name {    // RFC 7407
    prefix x509c2n;
    revision-date 2014-12-10;
  }

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

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

  description
    "This module contains a collection of YANG definitions for
    configuring NETCONF servers."
Copyright (c) 2014 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

revision "2015-02-02" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

feature ssh {
  description
    "The ssh feature indicates that the server supports the SSH transport protocol.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature tls {
  description
    "The tls feature indicates that the server supports the TLS transport protocol.";
  reference
    "RFC 5539: NETCONF over Transport Layer Security (TLS)";
}

feature listen {
  description
    "The listen feature indicates that the server supports opening a port to listen for incoming client connections.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)
RFC 5539: NETCONF over Transport Layer Security (TLS)";
}
feature call-home {
    description "The call-home feature indicates that the server supports connecting to the client";
    reference "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

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

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

    uses session-options-container;
    uses listen-container;
    uses call-home-container;
    uses ssh-container;
    uses tls-container;
}

grouping session-options-container {
    description "This grouping is used only to help improve readability of the YANG module.";
    container session-options {
        description "NETCONF session options, independent of transport or connection strategy.";

        leaf hello-timeout {
            type uint32 {
                range "0 | 10 .. 3600";
            }
            units "seconds";
        default '600';
        description "Specifies the number of seconds that a session may exist";
    }

Watsen & Schoenwaelder Expires August 6, 2015 [Page 14]
before the hello PDU is received. A session will be dropped if no hello PDU is received before this number of seconds elapses.

If this parameter is set to zero, then the server will wait forever for a hello message, and not drop any sessions stuck in ‘hello-wait’ state.

Setting this parameter to zero may permit denial of service attacks, since only a limited number of concurrent sessions may be supported by the server."

leaf idle-timeout {
  type uint32 {
    range "0 | 10 .. 360000";
  }
  units "seconds";
  default ’3600’;
  description
    "Specifies the number of seconds that a NETCONF session may remain idle without issuing any RPC requests. A session will be dropped if it is idle for an interval longer than this number of seconds. If this parameter is 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.

    This mechanism is independent of keep-alives, as it regards activity occurring at the NETCONF protocol layer, whereas the keep-alive mechanism regards transport-level activity.";
}

grouping listen-container {
  description
    "This grouping is used only to help improve readability of the YANG module.";
  container listen {
    description
      "Configures listen behavior";
    if-feature listen;
    leaf max-sessions {
      type uint16 {
        range "0 .. 1024";
      }
      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."

} 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 SSH and TLS transports.";
    case ssh {
      if-feature ssh;
      container ssh {
        description
        "SSH-specific listening configuration for inbound connections.";
        uses address-and-port-grouping {
          refine port {
            default 830;
          }
        }
        uses host-keys-container;
      }
    case tls {
      if-feature tls;
      container tls {
        description
        "TLS-specific listening configuration for inbound connections.";
        uses address-and-port-grouping {
          refine port {
            default 6513;
          }
        }
        uses certificates-container;
      }
    }
    uses keep-alives-container {
      refine keep-alives/interval-secs {

grouping call-home-container {
  description
  "This grouping is used only to help improve readability
  of the YANG module.";
  container call-home {
    if-feature call-home;
    description
    "Configures call-home behavior";
    list application {
      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;
          container ssh {
            description
            "Specifies SSH-specific call-home transport
            configuration.";
            uses endpoints-container {
              refine endpoints/endpoint/port {
                default 7777;
              }
            }
            uses host-keys-container;
          }
        }
        case tls {
          if-feature tls;
          container tls {
            description
            "Specifies TLS-specific call-home transport
            configuration.";
            uses endpoints-container {
              refine endpoints/endpoint/port {
                default 7777;
              }
            }
            uses host-keys-container;
          }
        }
      }
    }
  }
}

default 0; // disabled by default for listen connections
}
uses endpoints-container {
    refine endpoints/endpoint/port {
        default 8888;
    }
}
uses certificates-container;
}
}
container connection-type {
    description "Indicates the kind of connection to use.";
    choice connection-type {
        default persistent-connection;
        description "Selects between persistent and periodic connections.";
        case persistent-connection {
            container persistent {
                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.";
                uses keep-alives-container {
                    refine keep-alives/interval-secs {
                        default 15; // 15 seconds for call-home sessions
                    }
                }
            }
            case periodic-connection {
                container periodic {
                description "Periodically connect to NETCONF client, using the reconnection strategy, so the NETCONF client can deliver pending messages to the NETCONF server."
                For messages the NETCONF server wants to send to the NETCONF client, the NETCONF server should proactively connect to the NETCONF client, if not already, to send the messages immediately.";
                leaf timeout-mins {
                    type uint8;
units minutes;
default 5;
description
"The maximum amount of unconnected time the NETCONF server will wait until establishing a connection to the NETCONF client again. The NETCONF server MAY establish a connection before this time if it has data it needs to send to the NETCONF client. Note: this value differs from the reconnection strategy’s interval-secs value."
}
leaf linger-secs {
  type uint8;
  units seconds;
  default 30;
  description
  "The amount of time the NETCONF server should wait after last receiving data from or sending data to the NETCONF client’s endpoint before closing its connection to it. This is an optimization to prevent unnecessary connections."
}
}
}
container reconnect-strategy {
  description
  "The reconnection strategy guides how a NETCONF server reconnects to an NETCONF client, after losing a connection to it, even if due to a reboot. The NETCONF server starts with the specified endpoint and tries to connect to it count-max times, waiting interval-secs between each connection attempt, 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. NETCONF servers SHOULD support this flag 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. If no previous connection has
ever been established, last-connected defaults to
the first endpoint listed."
}
leaf interval-secs {
  type uint8;
  units seconds;
  default 5;
  description
  "Specifies the time delay between connection attempts
to the same endpoint. Note: this value differs from
the periodic-connection’s timeout-mins value."
}
leaf count-max {
  type uint8;
  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 ssh-container {
  description
  "This grouping is used only to help improve readability
  of the YANG module."
  container ssh {
    description
    "Configures SSH properties not specific to the listen
    or call-home use-cases"
    if-feature ssh;
    container x509 {
      if-feature ssh-x509-certs;
      uses trusted-certs-grouping;
    }
  }
}
grouping tls-container {
    description
        "This grouping is used only to help improve readability
         of the YANG module."
    container tls {
        description
            "Configures TLS properties for authenticating clients."
        if-feature tls;
        container client-auth {
            description
                "Container for TLS client authentication configuration."
            uses trusted-certs-grouping;
            container cert-maps {
                description
                    "The cert-maps container is used by a NETCONF server to
                   map the NETCONF client's presented X.509 certificate to a
                   NETCONF username. If no matching and valid cert-to-name
                   list entry can be found, then the NETCONF server MUST
                   close the connection, and MUST NOT accept NETCONF
                   messages over it."
                }
            }
        }
    }
}

grouping trusted-certs-grouping {
    description
        "This grouping is used by both the ssh and tls containers."
    container trusted-ca-certs {
        description
            "A list of Certificate Authority (CA) certificates that
             a NETCONF server can use to authenticate NETCONF client
             certificates. A client's certificate is authenticated
             if there is a chain of trust to a configured trusted CA
             certificate. The client certificate MAY be accompanied
             with additional certificates forming a chain of trust.
             The client's certificate is authenticated if there is
             path-validation from any of the certificates it presents
             to a configured trust anchor."
        leaf-list trusted-ca-cert {
            type binary;
            ordered-by system;
            nacm:default-deny-write;
            description
                "The binary certificate structure as specified by RFC
                 5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
                 "
            }
        }
    }
}

Watsen & Schoenwaelder   Expires August 6, 2015          [Page 21]
container trusted-client-certs {
  description
  "A list of client certificates that a NETCONF server can
  use to authenticate a NETCONF client’s certificate. A
  client’s certificate is authenticated if it is an exact
  match to a configured trusted client certificates.";
  leaf-list trusted-client-cert {
    type binary;
    ordered-by system;
    nacm:default-deny-write;
    description
    "The binary certificate structure, as
    specified by RFC 5246, Section 7.4.6, i.e.,:

    opaque ASN.1Cert<1..2^24>;

    ";
    reference
    "RFC 5246: The Transport Layer Security (TLS)
    Protocol Version 1.2";
  }
}

grouping host-keys-container {
  description
  "This grouping is used by both the listen and
  call-home containers";
  container host-keys {
    description
    "Parent container for the list of host-keys.";
    leaf-list host-key {
      type string;
      min-elements 1;
      ordered-by user;
      description
      "A user-ordered list of host-keys the SSH server
      considers when composing the list of server host
      key algorithms it will send to the client in its
      SSH_MSG_KEXINIT message. The value of the string
      is the unique identifier for a host-key configured
      on the system. How valid values are discovered is
      outside the scope of this module, but they are
      "RFC 5246: The Transport Layer Security (TLS)
      Protocol Version 1.2";
  }
}
grouping certificates-container {
  description
  "This grouping is used by both the listen and call-home containers";
  container certificates {
    description
    "Parent container for the list of certificates.";
    leaf-list certificate {
      type string;
      min-elements 1;
      description
      "An unordered list of certificates the TLS server can pick from when sending its Server Certificate message. The value of the string is the unique identifier for a certificate configured on the system. How valid values are discovered is outside the scope of this module, but they are envisioned to be the keys for a list of certificates provided by another YANG module";
      reference
      "RFC 5246: The TLS Protocol, Section 7.4.2";
    }
  }
}

grouping address-and-port-grouping {
  description
  "This grouping is used by both the ssh and tls containers for listen configuration.";
  leaf address {
    type inet:ip-address;
    description
    "The IP address of the interface to listen on.";
  }
  leaf port {
    type inet:port-number;
    description
    "The local port number on this interface the NETCONF server listens on.";
  }
}
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 the endpoint to connect to.";
            }
            leaf address {
                type inet:host;
                mandatory true;
                description
                    "The hostname or IP address or hostname of the endpoint.
                    If a hostname is provided and DNS resolves to more than
                    one IP address, the NETCONF server SHOULD try all of
                    the ones it can based on how its networking stack is
                    configured (e.g. v4, v6, dual-stack).";
            }
            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 not specified.";
            }
        }
    }
}

grouping keep-alives-container {
    description
        "This grouping is use by both listen and call-home configurations.";
    container keep-alives {
        description
            "Configures the keep-alive policy, to proactively test the
            aliveness of the NETCONF client.";
        reference
            "RFC VVVV: NETCONF Server and RESTCONF Server Configuration";
    }
}
4. The RESTCONF Server Configuration Model

4.1. Overview

4.1.1. The "listen" subtree
module: ietf-restconf-server
   +--rw restconf-server
      +--rw listen {listen}?
         +--rw max-sessions?   uint16
         +--rw endpoint* [name]
            +--rw name           string
      +--rw (transport)
         +--:(tls)
            +--rw address?        inet:ip-address
            +--rw port?           inet:port-number
            +--rw certificates
               +--rw certificate*   string
      +--rw keep-alives
         +--rw interval-secs?  uint8
         +--rw count-max?       uint8

The above subtree illustrates how the ietf-restconf-server YANG module enables configuration for listening for remote connections, as described in [draft-ietf-netconf-restconf]. Feature statements are used to limit both if listening is supported at all as well as for which transports. If listening for connections is supported, then the model enables configuring a list of listening endpoints, each configured with a user-specified name (the key field), the transport to use (i.e. TLS), and the IP address and port to listen on. The port field is optional, defaulting to the transport-specific port when not configured. Please see the YANG module (Section 4.2) for a complete description of these configuration knobs.

4.1.2. The "call-home" subtree
The above subtree illustrates how the ietf-restconf-server YANG module enables configuration for call home, as described in [draft-ietf-netconf-call-home]. Feature statements are used to limit both if call-home is supported at all as well as for which transports, if it is. If call-home is supported, then the model supports configuring a list of applications to connect to. Each application is configured with a user-specified name (the key field), the transport to be used (i.e. TLS), and a list of remote endpoints, each having a name, an IP address, and an optional port. Additionally, the configuration for each remote application indicates the connection-type (persistent vs. periodic) and associated parameters, as well as the reconnection strategy to use. Please see the YANG module (Section 4.2) for a complete description of these configuration knobs.
4.1.3. The "client-cert-auth" subtree

module: ietf-restconf-server
    +--rw restconf-server
        +--rw client-cert-auth {client-cert-auth}?
            ++--rw trusted-ca-certs
            |     +--rw trusted-ca-cert* binary
            ++--rw trusted-client-certs
            |     +--rw trusted-client-cert* binary
            ++--rw cert-maps
                +--rw cert-to-name* [id]
                    ++--rw id uint32
                    ++--rw fingerprint x509c2n:tls-fingerprint
                    ++--rw map-type identityref
                    ++--rw name string

The above subtree illustrates how the ietf-restconf-server YANG module enables configuration of client-certificate authentication. Specifically, this data-model provides 1) an ability to configure how client-certificates are authenticated and 2) how authenticated client-certificates are mapped to RESTCONF user names. Please see the YANG module (Section 4.2) for a complete description of these configuration knobs.

4.2. YANG Module

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

<CODE BEGINS> file "ietf-restconf-server@2015-02-02.yang"

module ietf-restconf-server {
    namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
    prefix "rcserver";

    import ietf-netconf-acm {
        prefix nacm;                     // RFC 6536
        revision-date 2012-02-22;
    }
    import ietf-inet-types {            // RFC 6991
        prefix inet;
        revision-date 2013-07-15;
    }
    import ietf-x509-cert-to-name {    // RFC 7407
        prefix x509c2n;
        revision-date 2014-12-10;
    }
}

Watsen & Schoenwaelder Expires August 6, 2015 [Page 28]
organization
"IETF NETCONF (Network Configuration) Working Group";

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

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

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

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

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

Copyright (c) 2014 IETF Trust and the persons identified as
authors of the code. All rights reserved.

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

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

revision "2015-02-02" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

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

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

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

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

    uses listen-container;
    uses call-home-container;
    uses client-cert-auth-container;
}

grouping listen-container {
    description
    "This grouping is used only to help improve readability
     of the YANG module.";
    container listen {
        description
        "Configures listen behavior";
        if-feature listen;
    }
}
leaf max-sessions {
    type uint16 {
        range "0 .. 1024";
    }
    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."
}
list endpoint {
    key name;
    description "List of endpoints to listen for RESTCONF connections on."
    leaf name {
        type string;
        description "An arbitrary name for the RESTCONF listen endpoint."
    }
    choice transport {
        mandatory true;
        description "Selects between available transports."
        case tls {
            container tls {
                description "TLS-specific listening configuration for inbound connections."
                uses address-and-port-grouping {
                    refine port {
                        default 443;
                    }
                }
            }
        }
        uses keep-alives-container {
            refine keep-alives/interval-secs {
                default 0; // disabled by default for listen connections
            }
        }
    }
}

grouping call-home-container {
    description
"This grouping is used only to help improve readability of the YANG module."

container call-home {
  if-feature call-home;
  description
    "Configures call-home behavior";
  list application {
    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 future transports augmented in."
    case tls {
      if-feature tls;
      container tls {
        description
          " Specifies TLS-specific call-home transport configuration.";
        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 9999;
          }
        }
        uses certificates-container;
      }
    }
  }
  container connection-type {
    description
      " Indicates the RESTCONF client’s preference for how the RESTCONF server’s connection is maintained."
    choice connection-type {
      default persistent-connection;
      description
        "Selects between persistent and periodic connections."
    case persistent-connection {
      container persistent {
        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."
uses keep-alives-container {
  refine keep-alives/interval-secs {
    default 15; // 15 seconds for call-home sessions
  }
}
}
case periodic-connection {
  container periodic {
    description "Periodically connect to RESTCONF client, using the reconnection strategy, so the RESTCONF client can deliver pending messages to the RESTCONF server.

    For messages the RESTCONF server wants to send to the RESTCONF client, the RESTCONF server should proactively connect to the RESTCONF client, if not already, to send the messages immediately."
    leaf timeout-mins {
      type uint8;
      units minutes;
      default 5;
      description "The maximum amount of unconnected time the RESTCONF server will wait until establishing a connection to the RESTCONF client again. The RESTCONF server MAY establish a connection before this time if it has data it needs to send to the RESTCONF client. Note: this value differs from the reconnection strategy’s interval-secs value."
    }
    leaf linger-secs {
      type uint8;
      units seconds;
      default 30;
      description "The amount of time the RESTCONF server should wait after last receiving data from or sending data to the RESTCONF client’s endpoint before closing its connection to it. This is an optimization to prevent unnecessary connections."
    }
  }
}

container reconnect-strategy {
  description
  "The reconnection strategy guides how a RESTCONF server reconnects to a RESTCONF client, after losing a connection to it, even if due to a reboot. The RESTCONF server starts with the specified endpoint and tries to connect to it count-max times, waiting interval-secs between each connection attempt, 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. RESTCONF servers SHOULD support this flag 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. If no previous connection has ever been established, last-connected defaults to the first endpoint listed.";
  leaf interval-secs {
    type uint8;
    units seconds;
    default 5;
    description
    "Specifies the time delay between connection attempts to the same endpoint. Note: this value differs from the periodic-connection’s timeout-mins value.";
  }
  leaf count-max {
    type uint8;
    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 client-cert-auth-container {
  description
  "This grouping is used only to help improve readability of the YANG module."
  container client-cert-auth {
    if-feature client-cert-auth;
    description
    "Container for TLS client certificate authentication configuration."
    container trusted-ca-certs {
      description
      "A list of Certificate Authority (CA) certificates that a NETCONF server can use to authenticate NETCONF client certificates. A client’s certificate is authenticated if there is a chain of trust to a configured trusted CA certificate. The client certificate MAY be accompanied with additional certificates forming a chain of trust. The client’s certificate is authenticated if there is path-validation from any of the certificates it presents to a configured trust anchor."
      leaf-list trusted-ca-cert {
        type binary;
        ordered-by system;
        nacm:default-deny-write;
        description
        "The binary certificate structure as specified by RFC 5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
        "
        reference
      }
    }
    container trusted-client-certs {
      description
      "A list of client certificates that a NETCONF server can use to authenticate a NETCONF client’s certificate. A client’s certificate is authenticated if it is an exact
match to a configured trusted client certificates.

leaf-list trusted-client-cert {
  type binary;
  ordered-by system;
  nacm:default-deny-write;
  description
    "The binary certificate structure, as
     specified by RFC 5246, Section 7.4.6, i.e.,:

     opaque ASN.1Cert<1..2^24>;

    ");
  reference
    "RFC 5246: The Transport Layer Security (TLS)
     Protocol Version 1.2";
}

container cert-maps {
  uses x509c2n:cert-to-name;
  description
    "The cert-maps container is used by a NETCONF server to
     map the NETCONF client’s presented X.509 certificate to a
     NETCONF username. If no matching and valid cert-to-name
     list entry can be found, then the NETCONF server MUST
     close the connection, and MUST NOT accept NETCONF
     messages over it.";
}


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

grouping address-and-port-grouping {
  description
  "This grouping is used by both the ssh and tls containers
  for listen configuration.";
  leaf address {
    type inet:ip-address;
    description
    "The IP address of the interface to listen on."
  }
  leaf port {
    type inet:port-number;
    description
    "The local port number on this interface the RESTCONF server
    listens on.";
  }
}


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 RESTCONF client.
       Defining more than one enables high-availability.";
      leaf name {
        type string;
        description
        "An arbitrary name for the endpoint to connect to.";
      }
      leaf address {
        type inet:host;
        mandatory true;
      }
    }
  }
}
description

"The hostname or IP address or hostname of the endpoint. If a hostname is provided and DNS resolves to more than one IP address, the RESTCONF server SHOULD try all of the ones it can based on how its networking stack is configured (e.g. v4, v6, dual-stack).";
}
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 not specified.";
}
}
}

grouping keep-alives-container {
  description
  "This grouping is use by both listen and call-home configurations."
  container keep-alives {
    description
    "Configures the keep-alive policy, to proactively test the aliveness of the RESTCONF client."
    reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models, Section 4"
    leaf interval-secs {
      type uint8;
      units seconds;
      description
      "Sets a timeout interval in seconds after which if no data has been received from the RESTCONF client, a message will be sent to request a response from the RESTCONF client. A value of '0' indicates that no keep-alive messages should be sent."
    }
    leaf count-max {
      type uint8;
      default 3;
      description
      "Sets the number of keep-alive messages that may be sent without receiving any data from the RESTCONF client before assuming the RESTCONF client is no longer alive. If this threshold is reached, the transport-level connection will be disconnected, which will trigger the reconnection strategy). The interval timer is reset after each transmission, thus an unresponsive RESTCONF client will
be dropped after approximately (count-max * interval-secs) seconds.
}
"
}
"

5. Implementation strategy for keep-alives

One of the objectives listed above, Keep-alives for persistent connections Section 2.6.6, indicates a need for a "keep-alive" mechanism. This section specifies how the keep-alive mechanism is to be implemented for both the SSH and TLS transports.

Both SSH and TLS have the ability to support keep-alives securely. Using the strategies listed below, the keep-alive messages are sent inside the encrypted tunnel and thus immune to attack.

5.1. Keep-alives for SSH

The SSH keep-alive solution that is expected to be used is ubiquitous in practice, though never being explicitly defined in an RFC. The strategy used is to purposely send a malformed request message with a flag set to ensure a response. More specifically, per section 4 of [RFC4253], either SSH peer can send a SSH_MSG_GLOBAL_REQUEST message with "want reply" set to ‘1’ and that, if there is an error, will get back a SSH_MSG_REQUEST_FAILURE response. Similarly, section 5 of [RFC4253] says that either SSH peer can send a SSH_MSG_CHANNEL_REQUEST message with "want reply" set to ‘1’ and that, if there is an error, will get back a SSH_MSG_CHANNEL_FAILURE response.

To ensure that the request will fail, current implementations of this keep-alive strategy (e.g. OpenSSH’s 'sshd' server) send an invalid "request name" or "request type", respectively. Abiding to the extensibility guidelines specified in Section 6 of [RFC4251], these implementations use the "name@domain". For instance, when configured to send keep-alives, OpenSSH sends the string "keepalive@openssh.com". In order to remain compatible with existing implementations, this draft does not require a specific "request name" or "request type" string be used, implementations are free to pick values of their choosing.
5.2. Keep-alives for TLS

The TLS keep-alive solution that is expected to be used is defined in [RFC6520]. This solution allows both peers to advertise if they can receive heartbeat request messages from its peer. For standard TLS connections, devices SHOULD advertise "peer_allowed_to_send", as per [RFC6520]. This advertisement is not a "MUST" in order to grandfather existing NETCONF/RESTCONF over TLS implementations. For NETCONF Call Home or RESTCONF Call Home, the network management system MUST advertise "peer_allowed_to_send" per [RFC6520]. This is a "MUST" so as to ensure devices can depend on it always being there for call home connections, which is when keep-alives are needed the most.

6. Security Considerations

The YANG modules defined in this memo are designed to be accessed via the NETCONF protocol [RFC6241]. Authorization for access to specific portions of conceptual data and operations within this module is provided by the NETCONF access control model (NACM) [RFC6536].

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

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

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

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

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

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

7. IANA Considerations

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

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

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

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

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

```yaml
name:         ietf-restconf-server
prefix:       rcserver
reference:    RFC VVVV
```

8. Other Considerations

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

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

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

10. References

10.1. Normative References


[draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-02 (work in progress), 2014.

[draft-ietf-netconf-restconf]

[draft-ietf-netconf-rfc5539bis]

10.2. Informative References

Appendix A. Examples

A.1. NETCONF Configuration using SSH Transport

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

```xml
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <session-options>
    <hello-timeout>600</hello-timeout>
    <idle-timeout>3600</idle-timeout>
  </session-options>
  <listen>
    <endpoint>
      <name>foo bar</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>my-rsa-key</host-key>
          <host-key>my-dss-key</host-key>
        </host-keys>
      </ssh>
    </endpoint>
  </listen>
  <call-home>
    <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>
  <ssh/>
</netconf-server>
```
A.2.  NETCONF Configuration using TLS Transport

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

```xml
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <session-options>
    <hello-timeout>600</hello-timeout>
    <idle-timeout>3600</idle-timeout>
  </session-options>
  <listen>
    <endpoint>
      <name>primary-netconf-endpoint</name>
      <tls>
        <address>11.22.33.44</address>
      </tls>
    </endpoint>
  </listen>
  <call-home>
    <application>
      <name>config-mgr</name>
    </application>
  </call-home>
</netconf-server>
```
<netconf-server>
  <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>
  <tls>
    <client-auth>
      <trusted-ca-certs>
        <trusted-ca-cert>
          QW4gRWFzdGVyIGVnZywgZm9yIHRoob3NlIHDobyBtaWdodCBsb29rICA6KQo=
        </trusted-ca-cert>
      </trusted-ca-certs>
      <trusted-client-certs>
        <trusted-client-cert>
          SSBhbSB0aGUgZ2FscnVzLCBnb28gZ29vIG1hbiwgdGhlIGVnZyBtZW4uCg==
        </trusted-client-cert>
        <trusted-client-cert>
          SSBhbSB0aGUgd2FscnVzLCB2YWdleSBoaGUgZ29vIG1hbiwgdGhlIGVnZyBtZW4uCg==
        </trusted-client-cert>
      </trusted-client-certs>
    </client-auth>
    <cert-maps>
      <cert-to-name>
        <id>1</id>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:specified</map-type>
      </cert-to-name>
      <cert-to-name>
        <id>2</id>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:san-any</map-type>
      </cert-to-name>
    </cert-maps>
  </tls>
</netconf-server>
A.3. RESTCONF Configuration using TLS Transport

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

```xml
<restconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">
  <listen>
    <endpoint>
      <name>primary-restconf-endpoint</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </endpoint>
  </listen>
  <call-home>
    <application>
      <name>config-mgr</name>
      <tls>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>55.66.77.88</address>
          </endpoint>
        </endpoints>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </application>
  </call-home>
</restconf-server>
```

Appendix B. Change Log

B.1. 00 to 01

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

B.2. 01 to 02

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

B.3. 02 to 03

- fixed tree diagrams and surrounding text

B.4. 03 to 04

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

B.5. 04 to 05

- Removed all refs to the old ietf-system-tls-auth module
- Removed YANG 1.1 style if-feature statements (loss some expressiveness)
- Removed the read-only (config false) lists of SSH host-keys and TLS certs
- Added an if-feature around session-options container
- Added ability to configure trust-anchors for SSH X.509 client certs
- Now imports by revision, per best practice
- Added support for RESTCONF server
- Added RFC Editor instructions

B.6. 05 to 06
- Removed feature statement on the session-options container (issue #21).
- Added NACM statements to YANG modules for sensitive nodes (issue #24).
- Fixed default RESTCONF server port value to be 443 (issue #26).
- Added client-cert-auth subtree to ietf-restconf-server module (issue #27).
- Updated draft-ietf-netmod-snmp-cfg reference to RFC 7407 (issue #28).
- Added description statements for groupings (issue #29).
- Added description for braces to tree diagram section (issue #30).
- Renamed feature from "rfc6187" to "ssh-x509-certs" (issue #31).

Appendix C. Open Issues

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

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net
Abstract

This document describes a YANG library, which provides information about all the YANG modules used by a device to represent management and protocol information. A YANG library can be shared by multiple protocols within the same device. Simple caching mechanisms are needed to allow clients to minimize retrieval of this information.

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 August 3, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

There is a need for standard mechanisms to identify the YANG modules and submodules that are in use by each server that utilizes YANG-based data abstraction. If a large number of YANG modules are utilized by the server, then the YANG library information needed can be relatively large. This information changes very infrequently, so it is important that clients be able to cache the YANG library and easily identify if their cache is out-of-date.

YANG library information can be different on every server, and can change at run-time or across a server reboot. Typically firmware upgrades are required to change the set of YANG modules used by a server.

The following information is needed by a client application (for each YANG module in the library) to fully utilize the YANG data modeling language.

- **name**: The mandatory YANG module name MUST be unique within a YANG library. All modules and submodules share the same namespace, including modules used for deviations.
1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- client
- server

1.1.2. YANG

The following terms are defined in [RFC6020]:

- module
- submodule

1.1.3. Terms

The following terms are used within this document:

- YANG library: a collection of YANG modules and submodules used by a server

1.1.4. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:
2. YANG Module Library

The "ietf-yang-library" module provides information about the YANG library used by a server.

YANG Tree Diagram for "ietf-yang-library" module:

```
+--ro modules
   +--ro module-set-id?   string
   +--ro module* [name revision]
      +--ro name         yang:yang-identifier
      +--ro revision     union
      +--ro schema?      inet:uri
      +--ro namespace    inet:uri
      +--ro feature*     yang:yang-identifier
      +--ro deviation*   yang:yang-identifier
      +--ro conformance  boolean
      +--ro submodules
         +--ro submodule* [name revision]
            +--ro name         yang:yang-identifier
            +--ro revision     union
            +--ro schema?      inet:uri
```

2.1. modules

This mandatory container holds the identifiers for the YANG data model modules supported by the server.

2.1.1. modules/module

This mandatory list contains one entry for each YANG data model module supported by the server. There MUST be an instance of this list for every YANG module that is used by the server.
2.2. YANG Library Module

The "ietf-yang-library" module defines monitoring information for the YANG modules used by a RESTCONF server.

The "ietf-yang-types" and "ietf-inet-types" modules from [RFC6991] are used by this module for some type definitions.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-yang-library@2015-01-30.yang"

module ietf-yang-library {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-library";
  prefix "yanglib";

  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>
    WG Chair: Mehmet Ersue
              <mailto:mehmet.ersue@nsn.com>
    WG Chair: Mahesh Jethanandani
              <mailto:mjethanandani@gmail.com>
    Editor:   Andy Bierman
              <mailto:andy@yumaworks.com>
    Editor:   Martin Bjorklund
              <mailto:mbj@tail-f.com>
    Editor:   Kent Watsen
              <mailto:kwatsen@juniper.net>";

  description
    "This module contains monitoring information about the YANG modules and submodules that are used within a YANG-based server."

  Copyright (c) 2015 IETF Trust and the persons identified as
authors of the code. All rights reserved.

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

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

// RFC Ed.: replace XXXX with actual RFC number and remove this note.
// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-yang-library-00.txt

// RFC Ed.: update the date below with the date of RFC publication and remove this note.
revision 2015-01-30 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: YANG Module Library.";
}

typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a specific date in YYYY-MM-DD format. TBD: make pattern more precise to exclude leading zeros.";
}

grouping module {
  description
    "The module data structure is represented as a grouping so it can be reused in configuration or another monitoring data structure.";

grouping common-leafs {
  description
    "Common parameters for YANG modules and submodules.";

  leaf name {
    type yang:yang-identifier;
  }
}
description "The YANG module or submodule name.";
}
leaf revision {
  type union {
    type revision-identifier;
    type string { length 0; }
  }
  description
    "The YANG module or submodule revision date.
      An empty string is used if no revision statement
      is present in the YANG module or submodule.";
}
leaf schema {
  type inet:uri;
  description
    "Contains a URL that represents the YANG schema
     resource for this module or submodule.

     This leaf will only be present if there is a URL
     available for retrieval of the schema for this entry.";
}
}

list module {
  key "name revision";
  description
    "Each entry represents one module currently
     supported by the server.";
  uses common-leafs;

  leaf namespace {
    type inet:uri;
    mandatory true;
    description
      "The XML namespace identifier for this module.";
  }

  leaf-list feature {
    type yang:yang-identifier;
    description
      "List of YANG feature names from this module that are
       supported by the server.";
  }

  leaf-list deviation {
    type yang:yang-identifier;
    description
      "List of YANG deviation module names used by this
       server to modify the conformance of the module
leaf conformance {
  type boolean;
  mandatory true;
  description
  "If 'true', then the server is claiming conformance to
  the YANG module identified in this entry.
  
  If 'false', then the server is not claiming any
  conformance for the YANG module identified by this
  entry. The module may be needed for reusable definitions
  such as extensions, features, identifies, typedefs,
  or groupings."
}

container submodules {
  description
  "Contains information about all the submodules used
  by the parent module entry";

  list submodule {
    key "name revision";
    description
    "Each entry represents one submodule within the
    parent module.";
    uses common-leafs;
  }
}
} // list module
} // grouping module

container modules {
  config false;
  description
  "Contains YANG module monitoring information.";

  leaf module-set-id {
    type string;
    description
    "Contains a server-specific identifier representing
    the current set of modules and submodules. The
    server MUST change the value of this leaf if the
    information represented by the 'module' list instances
    has changed.";
  }

  uses module;
}
3.  IANA Considerations

3.1.  YANG Module Registry

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

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

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

    name:         ietf-yang-library
    prefix:       yanglib
    // RFC Ed.: replace XXXX with RFC number and remove this note
    reference:    RFC XXXX

4.  Security Considerations

This section provides security considerations for the information that is defined by the "ietf-yang-library" module.

    [FIXME: follow template for sec. considerations]

5.  Normative References


Appendix A. Change Log

-- RFC Ed.: remove this section before publication.

A.1. draft-ietf-netconf-restconf-03 to 00

   o moved ietf-yang-library from RESTCONF draft to new draft

Appendix B. Open Issues

-- RFC Ed.: remove this section before publication.

   o conformance information: should ‘conformance’ leaf be removed and let real YANG conformance module augment as needed?

   o multi-protocol: should information be added to identify which protocols use each module or should each protocol define their own augmentations?

The YANG Library issue tracker can be found here:

https://github.com/netconf-wg/yang-library/issues

Authors’ Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net
Abstract

This document describes a method for applying patches to NETCONF datastores using data defined with the YANG data modeling language.

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 August 3, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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.
1. Introduction

There is a need for standard mechanisms to patch NETCONF [RFC6241] datastores which contain conceptual data that conforms to schema specified with YANG [RFC6020]. An "ordered edit list" approach is needed to provide client developers with a simpler edit request format that can be more efficient and also allow more precise client control of the transaction procedure than existing mechanisms.
This document defines a media type for a YANG-based editing mechanism that can be used with the HTTP PATCH method [RFC5789] or custom NETCONF operations (defined with the YANG rpc-stmt).

YANG Patch is designed to support multiple protocols with the same mechanisms. The RESTCONF protocol defined in [I-D.ietf-netconf-restconf] utilizes YANG Patch with the HTTP PATCH method. A new RPC operation can be defined to utilize YANG Patch in the NETCONF protocol. Both the RESTCONF and NETCONF protocols are designed to utilize the YANG data modeling language to specify content schema modules.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
- datastore
- configuration datastore
- protocol operation
- running configuration datastore
- server
- startup configuration datastore
- state data
- user
1.1.2. HTTP

The following terms are defined in [RFC2616]:

- entity tag
- fragment
- header line
- message body
- method
- path
- query
- request URI
- response body

1.1.3. YANG

The following terms are defined in [RFC6020]:

- container
- data node
- key leaf
- leaf
- leaf-list
- list
- presence container (or P-container)
- RPC operation (now called protocol operation)
- non-presence container (or NP-container)
- ordered-by system
- ordered-by user
1.1.4.  RESTCONF

The following terms are defined in [I-D.ietf-netconf-restconf]:

- data resource
- datasource resource
- patch
- RESTCONF capability
- target resource

1.1.5.  Terms

The following terms are used within this document:

- YANG Patch: a conceptual edit request using the "yang-patch" YANG container, defined in Section 3. In HTTP, refers to a PATCH method where the media type is "application/yang.patch+xml" or "application/yang.patch+json".

- YANG Patch Status: a conceptual edit status response using the YANG "yang-patch-status" container, defined in Section 3. In HTTP, refers to a response message for a PATCH method, where the message body is identified by the media type "application/yang.patch-status+xml" or "application/yang.patch-status+json".

1.1.6.  Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node and "*" denotes a "list" and "leaf-list".
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.
2. YANG Patch

A "YANG Patch" is an ordered list of edits that are applied to the target datastore by the server. The specific fields are defined with the ‘application/yang.patch’ extension definition in the YANG module Section 3.

For RESTCONF, the YANG Patch operation is invoked by the client by sending a PATCH method request with the YANG Patch media type. A message body representing the YANG Patch input parameters MUST be provided.

The RESTCONF server MUST return the Accept-Patch header in an OPTIONS response, as specified in [RFC5789], which includes the media type for YANG Patch.

Example:

Accept-Patch: application/yang.patch

2.1. Target Resource

The YANG Patch operation uses a conceptual root within a NETCONF configuration datastore to identity the patch point for the edit operation. This root can be the datastore itself, or 1 or more data nodes within the datastore.

For RESTCONF, the target resource is derived from the request URI.

For NETCONF, the target resource MUST be defined as an input parameter in the YANG "rpc" statement.

2.2. yang-patch Input

A data element representing the YANG Patch is sent by the client to specify the edit operation request. When used with the HTTP PATCH method, this data is identified by the YANG Patch media type.

YANG Tree Diagram For "yang-patch" Container
2.3. yang-patch-status Output

A data element representing the YANG Patch Status is returned to the client to report the detailed status of the edit operation. When used with the HTTP PATCH method, this data is identified by the YANG Patch Status media type, and the syntax specification is defined by the ‘application/yang.patch-status’ extension statement defined in Section 3.

YANG Tree Diagram For "yang-patch-status" Container:

```
+--rw yang-patch-status
  +--rw patch-id?   string
  +--rw (global-status)?
    |  +--:(global-errors)
    |     +--ro errors
    |  +--:(ok)
    |     +--rw ok?              empty
  +--rw edit-status
    +--rw edit [edit-id]
      +--rw edit-id     string
      +--rw (edit-status-choice)?
        +--:(ok)
        |     +--rw ok?         empty
        +--:(errors)
    +--rw value
```

2.4. Target Data Node

The target data node for each edit operation is determined by the value of the target resource in the request and the "target" leaf within each "edit" entry.

If the target resource specified in the request URI identifies a datastore resource, then the path string in the "target" leaf is an
absolute path expression. The first node specified in the "target" leaf is a top-level data node defined within a YANG module.

If the target resource specified in the request URI identifies a data resource, then the path string in the "target" leaf is a relative path expression. The first node specified in the "target" leaf is a child node of the data node associated with the target resource.

2.5. Edit Operations

Each YANG patch edit specifies one edit operation on the target data node. The set of operations is aligned with the NETCONF edit operations, but also includes some new operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>create a new data resource if it does not already exist or error</td>
</tr>
<tr>
<td>delete</td>
<td>delete a data resource if it already exists or error</td>
</tr>
<tr>
<td>insert</td>
<td>insert a new user-ordered data resource</td>
</tr>
<tr>
<td>merge</td>
<td>merge the edit value with the target data resource; create if it does not already exist re-order the target data resource</td>
</tr>
<tr>
<td>move</td>
<td>re-order the target data resource</td>
</tr>
<tr>
<td>replace</td>
<td>replace the target data resource with the edit value</td>
</tr>
<tr>
<td>remove</td>
<td>remove a data resource if it already exists or no error</td>
</tr>
</tbody>
</table>

YANG Patch Edit Operations

2.6. Error Handling

If a well-formed, schema-valid YANG Patch message is received, then the server will process the supplied edits in ascending order. The following error modes apply to the processing of this edit list:

All the specified edits MUST be applied or the target datastore contents SHOULD be returned to its original state before the PATCH method started. The server MAY fail to restore the contents of the target datastore completely and with certainty. It is possible for a rollback to fail or an "undo" operation to fail.

The server will save the running datastore to non-volatile storage if it has changed, after the edits have been attempted.
2.7. yang-patch RESTCONF Capability

A URI is defined to identify the YANG Patch extension to the base
RESTCONF protocol. If the server supports the YANG Patch media type,
then the "yang-patch" RESTCONF capability defined in Section 4.4 MUST
be present in the "capability" leaf-list in the
"ietf-restconf-monitoring" module defined in
[I-D.ietf-netconf-restconf].

3. YANG Module

The "ietf-yang-patch" module defines conceptual definitions with the
'restconf-media-type' extension statements, which are not meant to be
implemented as datastore contents by a server.

The "ietf-restconf" module from [I-D.ietf-netconf-restconf] is used
by this module for the 'restconf-media-type' extension definition.

RFC Ed.: update the date below with the date of RFC publication and
remove this note.

<CODE BEGINS> file "ietf-yang-patch@2015-01-24.yang"

module ietf-yang-patch {
  prefix "ypatch";

  import ietf-restconf {
    prefix rc;
    revision-date 2015-01-30;
  }

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

This module contains conceptual YANG specifications for the YANG Patch and YANG Patch Status data structures.

Note that the YANG definitions within this module do not represent configuration data of any kind. The YANG grouping statements provide a normative syntax for XML and JSON message encoding purposes.

Copyright (c) 2015 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-yang-patch-03.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-01-30 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: YANG Patch Media Type.";
}

typedef target-resource-offset {
  type string {
    length "1 .. max";
  }
  description
"Contains a relative Data Resource Identifier formatted string to identify a specific data sub-resource instance. The document root for all data resources is a target data resource that is specified in the object definition using this data type.";

rc:restconf-media-type "application/yang.patch" {
  uses yang-patch;
}

rc:restconf-media-type "application/yang.patch-status" {
  uses yang-patch-status;
}

grouping yang-patch {

description
  "A grouping that contains a YANG container representing the syntax and semantics of a YANG Patch edit request message."

container yang-patch {
  description
    "Represents a conceptual sequence of datastore edits, called a patch. Each patch is given a client-assigned patch identifier. Each edit MUST be applied in ascending order, and all edits MUST be applied. If any errors occur, then the target datastore MUST NOT be changed by the patch operation.

    A patch MUST be validated by the server to be a well-formed message before any of the patch edits are validated or attempted.

    YANG datastore validation (defined in RFC 6020, section 8.3.3) is performed after all edits have been individually validated.

    It is possible for a datastore constraint violation to occur due to any node in the datastore, including nodes not included in the edit list. Any validation errors MUST be reported in the reply message.";

  reference
    "RFC 6020, section 8.3.";

  leaf patch-id {

type string;
description
"An arbitrary string provided by the client to identify
the entire patch. This value SHOULD be present in any
audit logging records generated by the server for the
patch. Error messages returned by the server pertaining
to this patch will be identified by this patch-id value.";
}

leaf comment {
type string {
  length "0 .. 1024";
}
description
"An arbitrary string provided by the client to describe
the entire patch. This value SHOULD be present in any
audit logging records generated by the server for the
patch.";
}

list edit {
  key edit-id;
  ordered-by user;

description
"Represents one edit within the YANG Patch
request message. The edit list is applied
in the following manner:

- The first edit is conceptually applied to a copy
  of the existing target datastore, e.g., the
  running configuration datastore.
- Each ascending edit is conceptually applied to
  the result of the previous edit(s).
- After all edits have been successfully processed,
  the result is validated according to YANG constraints.
- If successful, the server will attempt to apply
  the result to the target datastore. ";

leaf edit-id {
type string;
description
"Arbitrary string index for the edit.
Error messages returned by the server pertaining
to a specific edit will be identified by this
value.";
}
leaf operation {
  type enumeration {
    enum create {
      description "The target data node is created using the supplied value, only if it does not already exist.";
    }
    enum delete {
      description "Delete the target node, only if the data resource currently exists, otherwise return an error.";
    }
    enum insert {
      description "Insert the supplied value into a user-ordered list or leaf-list entry. The target node must represent a new data resource.";
    }
    enum merge {
      description "The supplied value is merged with the target data node.";
    }
    enum move {
      description "Move the target node. Reorder a user-ordered list or leaf-list. The target node must represent an existing data resource.";
    }
    enum replace {
      description "The supplied value is used to replace the target data node.";
    }
    enum remove {
      description "Delete the target node if it currently exists.";
    }
  }
  mandatory true;
  description "The datastore operation requested for the associated edit entry";
}
leaf target {
  type target-resource-offset;
mandatory true;
description
"Identifies the target data resource for the edit
operation."
}

leaf point {
when "return (operation = 'insert' or " +
"./operation = 'move') and " +
"(./where = 'before' or ./where = 'after')" {

description
"Point leaf only applies for insert or move
operations, before or after an existing entry."
}
type target-resource-offset;
description
"The absolute URL path for the data node that is being
used as the insertion point or move point for the
target of this edit entry."
}

leaf where {
when "return (operation = 'insert' or../operation = 'move')" {

description
"Where leaf only applies for insert or move
operations."
}
type enumeration {
enum before {

description
"Insert or move a data node before the data resource
identified by the 'point' parameter."
}
enum after {

description
"Insert or move a data node after the data resource
identified by the 'point' parameter."
}
enum first {

description
"Insert or move a data node so it becomes ordered
as the first entry."
}
enum last {

description
"Insert or move a data node so it becomes ordered
as the last entry."
}
default last;
description "Identifies where a data resource will be inserted or moved. YANG only allows these operations for list and leaf-list data nodes that are ordered-by user.";
}

anyxml value {
  when "(.operation = 'create' or " + 
    ".operation = 'merge' " + 
  "or .operation = 'replace' or " + 
    ".operation = 'insert')" {
    description "Value node only used for create, merge, replace, and insert operations";
  }
}

description "Value used for this edit operation. The anyxml value MUST represent a container with exactly one child node, which MUST identify the target resource associated with the 'target' leaf.

For example, suppose the target node is a YANG container named foo:

    container foo {
      leaf a { type string; }
      leaf b { type int32; }
    }

The value node will contain one instance of foo:

    <value>
    <foo xmlns='example-foo-namespace'>
      <a>some value</a>
      <b>42</b>
    </foo>
    
    </value>
grouping yang-patch-status {

description
  "A grouping that contains a YANG container
  representing the syntax and semantics of
  YANG Patch status response message.";
}

container yang-patch-status {
  description
  "A container representing the response message
  sent by the server after a YANG Patch edit
  request message has been processed.";

  leaf patch-id {
    type string;
    description
    "The patch-id value used in the request";
  }

  choice global-status {
    description
    "Report global errors or complete success.
    If there is no case selected then errors
    are reported in the edit-status container.";

    case global-errors {
      uses rc:errors;
      description
      "This container will be present if global
      errors unrelated to a specific edit occurred.";
    }

    leaf ok {
      type empty;
      description
      "This leaf will be present if the request succeeded
      and there are no errors reported in the edit-status
      container.";
    }
  }

  container edit-status {
    description
    "This container will be present if there are
    edit-specific status responses to report.
    If all edits succeeded and the 'global-status'
    returned is 'ok', then a server MAY omit this
    container";
  }
}
list edit {
  key edit-id;

description
  "Represents a list of status responses,
corresponding to edits in the YANG Patch
request message. If an edit entry was
skipped or not reached by the server,
then this list will not contain a corresponding
entry for that edit.";

leaf edit-id {
  type string;
  description
    "Response status is for the edit list entry
    with this edit-id value.";
}

choice edit-status-choice {
  description
    "A choice between different types of status
    responses for each edit entry.";
  leaf ok {
    type empty;
    description
      "This edit entry was invoked without any
      errors detected by the server associated
      with this edit.";
  }
  case errors {
    uses rc:errors;
    description
      "The server detected errors associated with the
      edit identified by the same edit-id value.";
  }
}

} // grouping yang-patch-status

<CODE ENDS>
4. IANA Considerations

4.1. YANG Module Registry

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

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

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

name: ietf-yang-patch
prefix: ypatch
reference: RFC XXXX

4.2. application/yang.patch Media Types

The MIME media type for a YANG Patch document is application/yang.patch.

Type name: application
Subtype name: yang.patch
Required parameters: TBD
Optional parameters: TBD
Encoding considerations: TBD
Security considerations: TBD
Interoperability considerations: TBD

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

4.3. application/yang.patch-status Media Types

The MIME media type for a YANG Patch status document is application/yang.patch-status.

Type name: application

Subtype name: yang.patch-status

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

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

4.4. RESTCONF Capability URNs

This document registers several capability identifiers in "RESTCONF Protocol Capability URNs" registry

<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>:yang-patch</td>
</tr>
<tr>
<td>urn:ietf:params:restconf:capability:yang-patch:1.0</td>
</tr>
</tbody>
</table>

5. Security Considerations

The YANG Patch media type does not introduce any significant new security threats, beyond what is described in [I-D.ietf-netconf-restconf]. This document defines edit processing instructions for a variant of the PATCH method, as used within the RESTCONF protocol.

It is important for server implementations to carefully validate all the edit request parameters in some manner. If the entire YANG Patch request cannot be completed, then no configuration changes to the system are done.

A server implementation SHOULD attempt to prevent system disruption due to partial processing of the YANG Patch edit list. It may be possible to construct an attack on such a server, which relies on the edit processing order mandated by YANG Patch.
6. Normative References

[I-D.ietf-netconf-restconf]


[W3C.REC-xml-20081126]

Appendix A. Acknowledgements

The authors would like to thank the following people for their contributions to this document: Rex Fernando.

Appendix B. Change Log

-- RFC Ed.: remove this section before publication.
B.1.  02 to 03

- added usage of restconf-media-type extension to map the yang-patch
  and yang-patch-status groupings to media types
- added yang-patch RESTCONF capability URI
- Added sub-section for terms used from RESTCONF
- filled in security considerations section

B.2.  01 to 02

- Reversed order of change log
- Clarified anyxml structure of "value" parameter within a YANG
  patch request (github issue #1)
- Updated RESTCONF reference
- Added note to open issues section to check github instead

B.3.  00 to 01

- Added text requiring support for Accept-Patch header, and removed
  ‘Identification of YANG Patch capabilities’ open issue.
- Removed ‘location’ leaf from yang-patch-status grouping
- Removed open issue ‘Protocol independence’ because the location
  leaf was removed.
- Removed open issue ‘RESTCONF coupling’ because there is no concern
  about a normative reference to RESTCONF. There may need to be a
  YANG 1.1 mechanism to allow protocol template usage (instead of
  grouping wrapper).
- Removed open issue ‘Is the delete operation needed’. It was
decided that both delete and remove should remain as operations
and clients can choose which one to use. This is not an
implementation burden on the server.
- Removed open issue ‘global-errors needed’. It was decided that
they are needed as defined because the global <ok/> is needed and
the special key value for edit=global error only allows for 1
global error.
- Removed open issue ‘Is location leaf needed’. It was decided that it is not needed so this leaf has been removed.

- Removed open issue ‘Bulk editing support in yang-patch-status’. The ‘location’ leaf has been removed so this issue is no longer applicable.

- Removed open issue ‘Edit list mechanism’. Added text to the ‘edit’ list description-stmt about how the individual edits must be processed. There is no concern about duplicate edits which cause intermediate results to be altered by subsequent edits in the same edit list.

B.4. bierman:yang-patch-00 to ietf:yang-patch-00

- Created open issues section

Appendix C. Open Issues

-- RFC Ed.: remove this section before publication.

Refer to the github issue tracker for any open issues:

https://github.com/netconf-wg/yang-patch/issues

Appendix D. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface. The "example-jukebox" YANG module is defined in [I-D.ietf-netconf-restconf].

YANG Tree Diagram for "example-jukebox" Module:
D.1. YANG Patch Examples

This section includes RESTCONF examples. NETCONF examples are TBD. Most examples are shown in JSON encoding [RFC7158], and some are shown in XML encoding [W3C.REC-xml-20081126].

D.1.1. Add Resources: Error

The following example shows several songs being added to an existing album. Each edit contains one song. The first song already exists, so an error will be reported for that edit. The rest of the edits were not attempted, since the first edit failed.
Request from client:

PATCH /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{
    "ietf-yang-patch:yang-patch": {
        "patch-id": "add-songs-patch",
        "edit": [
            {
                "edit-id": 1,
                "operation": "create",
                "target": "/song",
                "value": {
                    "song": {
                        "name": "Bridge Burning",
                        "location": "/media/bridge_burning.mp3",
                        "format": "MP3",
                        "length": 288
                    }
                }
            },
            {
                "edit-id": 2,
                "operation": "create",
                "target": "/song",
                "value": {
                    "song": {
                        "name": "Rope",
                        "location": "/media/rope.mp3",
                        "format": "MP3",
                        "length": 259
                    }
                }
            },
            {
                "edit-id": 3,
                "operation": "create",
                "target": "/song",
                "value": {
                    "song": {
                        "name": "Dear Rosemary",
                        "location": "/media/dear_rosemary.mp3",
                        "format": "MP3",
                        "length": 269
                    }
                }
            }
        ]
    }
}
Response from server:

HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json

```
{
  "ietf-yang-patch:yang-patch-status" : {
    "patch-id" : "add-songs-patch",
    "edit-status" : {
      "edit" : [
        {
          "edit-id" : 1,
          "errors" : {
            "error" : [
              {
                "error-type": "application",
                "error-tag": "data-exists",
                "error-path": "/example-jukebox:jukebox/library
                             /artist=Foo%20Fighters/album=Wasting%20Light
                             /song=Burning%20Light",
                "error-message": "Data already exists, cannot be created"
              }
            ]
          }
        }
      ]
    }
  }
}
```

D.1.2. Add Resources: Success

The following example shows several songs being added to an existing album.

- Each of 2 edits contains one song.
Both edits succeed and new sub-resources are created

Request from client:

PATCH /restconf/data/example-jukebox:jukebox/
   library/artist=Foo%20Fighters/album=Wasting%20Light
HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{
   "ietf-yang-patch:yang-patch": {
      "patch-id": "add-songs-patch-2",
      "edit": [
         {
            "edit-id": 1,
            "operation": "create",
            "target": "/song",
            "value": {
               "song": {
                  "name": "Rope",
                  "location": "/media/rope.mp3",
                  "format": "MP3",
                  "length": 259
               }
            }
         },
         {
            "edit-id": 2,
            "operation": "create",
            "target": "/song",
            "value": {
               "song": {
                  "name": "Dear Rosemary",
                  "location": "/media/dear_rosemary.mp3",
                  "format": "MP3",
                  "length": 269
               }
            }
         }
      ]
   }
}

Response from server:

HTTP/1.1 200 Success
D.1.3. Move list entry example

The following example shows a song being moved within an existing playlist. Song "1" in playlist "Foo-One" is being moved after song "3" in the playlist. The operation succeeds, so a non-error reply example can be shown.
Request from client:

PATCH /restconf/data/example-jukebox:jukebox/
   playlist=Foo-One HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{  
   "ietf-yang-patch:yang-patch" : {  
      "patch-id" : "move-song-patch",
      "comment" : "Move song 1 after song 3",
      "edit" : [  
         {  
            "edit-id" : 1,
            "operation" : "move",
            "target" : "/song/1",
            "point" : "/song3",
            "where" : "after"
         }
      ]
   }
}

Response from server:

HTTP/1.1 400 OK
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json

{  
   "ietf-restconf:yang-patch-status" : {  
      "patch-id" : "move-song-patch",
      "ok" : [null]
   }
}

Authors’ Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com
Martin Bjorklund
Tail-f Systems
Email: mbj@tail-f.com

Kent Watsen
Juniper Networks
Email: kwatsen@juniper.net
Zero Touch Provisioning for NETCONF Call Home (ZeroTouch)
draft-ietf-netconf-zerotouch-02

Abstract

This draft presents a technique for establishing a secure NETCONF connection between a newly deployed IP-based device, configured with just its factory default settings, and its rightful owner’s network management system (NMS).

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents 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
1. Introduction

A fundamental business requirement 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.
The solution presented herein enables a device to securely obtain a bootstrapping configuration from the network without any operator input. Significantly, this configuration may configure the device to securely call home using NETCONF Call Home [draft-ietf-netconf-call-home].

Central to this solution is the device being able to process a set of files locally, without any need to reach out to the network again. As consequence, how the files are obtained is not critical to the security of the solution. The files can be read over any networking layer or medium. By example, the files could be loaded using a USB flash drive physically plugged into a device.

The solution presented below focuses on supporting IP networks that may have a DHCP server. Solutions for other deployment scenarios may be defined by drafts in the future.

1.1. Use Cases

- Connecting to a remotely administered network
  
  This use-case involves scenarios, such as a remote branch office or convenience store, whereby the device connects an access gateway device to an ISP’s network. In this case, the device receives only generic networking settings provided by the ISP’s DHCP server, with no site-specific customizations possible. In such a case, the device has no recourse but to reach out to the public Internet its initial configuration.

- Connecting to a locally administered network
  
  This use-case covers all other scenarios and differs only in that the device may additionally receive site-specific information from the network, which could direct it to use a local server for its initial configuration. If no site-specific information is provided, or the device is unable to use the information provided, it can then reach out to network just as it would for a remotely administered network.

1.2. Terminology

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].
1.3. Tree Diagrams

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

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

2. High-level Design

2.1. Design Overview

The following diagram illustrates the overall solution presented in this draft. Note that some of the interactions illustrated below occur at different times, only the numbered interactions (1-3) occur at the time a device is bootstrapping itself.
The boxes in this diagram are described next. A sequence diagram explaining the various calls follows in Section 2.2.

- **Vendor**

  Vendors manufacture the devices supporting NETCONF ZeroTouch. To support this solution, Vendors must support a one-time enrollment process per business organization owning the NMS. Vendors must also support sending additional information to the business organization about the devices that have been shipped for device orders it places.

- **Device**

  The devices supporting NETCONF ZeroTouch will only attempt the bootstrapping process when booting with its factory default
configuration. As illustrated above, the bootstrapping process consists of three interactions:

1. When joining the network, the device will attempt to configure IP networking from a DHCP server. If the device is able to reach a DHCP server, it may discover additional bootstrapping information. The additional bootstrapping information consists of one or more additional Bootstrap Servers the device should try to connect to.

2. The device sequentially processes its list of Bootstrap Servers, prioritizing any that might have been learned from the DHCP server. Once the device has successfully configured itself using the bootstrapping information, it notifies the bootstrapping server for monitoring purposes.

3. Assuming the bootstrapping information configures the device appropriately, the device will initiate a NETCONF Call Home connection [draft-ietf-netconf-call-home].

More information about Devices is in Section 4.

- DHCP Server

  This draft assumes the use of a DHCP server but, in reality, the solution is not intrinsically tied to using a DHCP server. Any mechanism or combination of mechanisms that can provide dynamic networking assignment would equally do.

  Assuming the use of DHCP, this draft defines a specific DHCP Option for the discovering of additional bootstrapping information. More information about the ZeroTouch DHCP Option is in Section 7.1.

- Bootstrap Server

  Bootstrap Servers host the bootstrapping information staged by NMSs for the devices to find. The Bootstrap Server presents a simple REST interface for devices to obtain both their bootstrapping information as well as notify the Bootstrapping Server when it has successfully completed the bootstrapping process.

  Bootstrap Servers may be deployed on the public Internet or on a local network. Devices may be preconfigured with a list of well-known Bootstrap Servers. Additional Bootstrap Servers (i.e. not in the device’s preconfigured list) must be discovered from a DHCP server.
How Bootstrap Servers are deployed is out of scope of this draft, but there are a couple points worth noting. Firstly, it is expected that Internet based Bootstrap Servers will initially be hosted by Vendors, whilst waiting for 3rd-party servers to become available. Secondly, it is expected that locally administrated networks with in-house solutions might bundle the Bootstrap Server into another system (e.g., the NMS), where having the features integrated can streamline various workflows.

More information about Bootstrap Servers is in Section 3.

o Network Management System

The NMS is a term used here loosely to represent any system, or collection of systems, deployed by a business organization to manage its devices. An NMS being able to establish a secure NETCONF connection with devices purchased by its organization is the ultimate goal of this solution presented by this draft. More information about the Network Management System is in Section 5.

2.2. Interactions

The following diagram illustrates the interactions between the entities described in the previous section. Note that the interactions can be roughly categorized as those that occur before a device powers on and those that occur after a device powers on.
1. imports trust anchor
<-------------------------------
2. signs up for owner cert
<-------------------------------
3. orders devices
<-------------------------------
4. ships
<-------------------------------
5. provides serial-numbers and/or IDevID cert(s), and ownership vouchers

<-------------------------------
6. stage bootstrap data
<-------------------------------
7. stage bootstrap info (optional)
<-------------------------------
8. power on
<-------------------------------
9. get networking settings and staged bootstrap info (if any)
<-------------------------------
10. update boot-image, if needed, and install config, if valid
<-------------------------------
11. netconf
    call-home

These interactions are described below.

1. An organization, upon deciding to deploy a Vendor’s devices for NETCONF ZeroTouch, would import into its NMS the IDevID trust anchor certificate from the Vendor. This certificate is later used by the NMS to authenticate device identities during NETCONF call home connections.

2. An organization needs to sign up to a Vendor-provided ZeroTouch program. This program entails the Vendor providing a signed Owner certificate to the organization (depicted here), as well as a commitment to sign Ownership Vouchers for future device orders (interaction #5).

3. Subsequently, the organization may place orders to the Vendor for devices supporting ZeroTouch. The ordering process may entail an explicit request for ZeroTouch support, as the Vendor providing the files in step #5 may not be enabled by default.

4. The Vendor ships the devices to the various addresses specified in the device order. For example, to an organization’s inventory warehouse, where the devices are stored in batches to supply internal requests. In another example, the devices may be shipped to their final deployment destinations.

5. In order to support ZeroTouch, the Vendor sends to the organization information about the devices it shipped. This information may be sent to the organization via email or a portal site. The information includes the serial-number and/or IDevID certificate, for each device, as well as one more Ownership Vouchers, assigning ownership for the devices to the organization.

6. In anticipation for the devices performing the ZeroTouch process, the NMS configures the Bootstrap Server. This configuration includes everything a device needs to securely connect to the NMS.

7. For deployments where the DHCP server can be customized, the NMS may configure the DHCP server to provide the device a list of additional Bootstrap Servers to consider, in addition to those the device knows of by default. This customization can be configured at a global level in the DHCP server, as it is not dependent on the type of device in any way.

8. At some point, the device powers on and, when having its factory default configuration, initiates the ZeroTouch process.
9. The device obtains from the DHCP server a dynamic network assignment. The device may also at this time discover a list of additional bootstrap servers, as optionally configured by the NMS in step #7.

10. The device iterates over its list of Bootstrap Servers, until it can successfully bootstrap its initial configuration. If it is unable to bootstrap an initial configuration, the device boots as normal. If the staged information directs the device to load a new image, it does so and reboots. If the device reboots, it continues to have a factory default configuration state, which then bring it back to this state, when it would then have the correct image. The device then loads the staged configuration into its running datastore, after validating that the configuration was signed by its rightful owner, as designated by the Ownership Voucher.

11. Assuming the bootstrapping information configures the device appropriately, the device will initiate a NETCONF Call Home [draft-ietf-netconf-call-home] connection to the NMS, which then takes over the on-going management of the device.

3. Bootstrap Server

A Bootstrap Server MUST implement the southbound interface defined below. In order to support the southbound interface, the Bootstrap Server will also need to have a northbound interface, which is described in general terms below.

3.1. Northbound Interface

The Bootstrap Server will need to provide a northbound interface of some sort to enable configuration of the bootstrapping information for the devices. Defining this interface is out of scope for this document, but it northbound interface is generally expected to:

- Enable listing, creation, modification, and deletion of entries
- Enable determining a device’s current bootstrapping state
- Enable alerting external systems when a device sends notifications

3.2. Southbound Interface

The Bootstrap Server’s southbound interface is a REST API that is described by the YANG [RFC6020] module defined in this section and presented using RESTCONF [draft-ietf-netconf-restconf]. Example usage of this API is provided in Appendix A.2.
A tree diagram describing the Bootstrap Server’s southbound interface follows:

module: ietf-zerotouch-bootstrap-server
   +--ro devices
      +--ro device* [unique-id]
         +--ro unique-id            string
         +--ro ownership-voucher
            |   +--ro voucher       binary
            |   +--ro issuer-crl?   string
         +--ro owner-certificate
            |   +--ro certificate    string
            |   +--ro issuer-crl?    string
         +--ro boot-image!
            |   +--ro name         string
            |   +--ro path         string
            |   +--ro signature    string
         +--ro configuration
            +--ro config
            +--ro signature    string

rpcs:
   +----x notification
      +----w input
         +----w unique-id string
         +----w type         enumeration
         +----w message?    string

In the above diagram, notice that the entire data model is read-only, as devices can only pull data from the Bootstrap Server. The data model also defines a single RPC, which is used by the device to provide asynchronous notifications.

The Bootstrap Server’s southbound interface is normatively defined by the following YANG module:

<CODE BEGINS> file "ietf-zerotouch-bootstrap-server@2015-03-09.yang"

module ietf-zerotouch-bootstrap-server {

namespace
prefix "ztbs";

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

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

description
"This module defines the southbound interface for ZeroTouch Bootstrap 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 "2015-03-09" {
  description
    "Initial version";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF Call Home";
}

// top-level container
container devices {
  config false;
  description
    "A list of device entries";
  list device {
    key unique-id;
    leaf unique-id {
      type string;
    }
  }

  container ownership-voucher {
    description
      "This container contains the Ownership Voucher that the

device uses to ascertain the identity of its rightful owner, as certified by its Vendor.

leaf voucher {
  type binary;
  mandatory true;
  description
    "A Vendor-specific encoding binding unique device identifiers to an owner identifier value matching the value encoded in the owner-certificate below. An example format for a voucher is presented in the Appendix of RFC XXXX."
}

leaf issuer-crl {
  type string;
  description
    "An absolute path to a CRL for the issuer used by the Vendor to sign Ownership Vouchers. The CRL should be as up to date as possible. This leaf is optional as it is primarily to support deployments where the device is unable to download the CRL from the CRL distribution point URLs listed in the Vendor’s trust anchor certificate."
}

container owner-certificate {
  description
    "It is intended that the device will fetch this container as a whole, as it contains values that need to be processed together."
  leaf certificate {
    type string;
    mandatory true;
    description
      "This is an X.509 certificate, signed by a Vendor, for a business organization. This certificate must encode a Vendor-assigned value identifying the organization. This identifier must match the owner identifier encoded in the Ownership Voucher."
  }
  leaf issuer-crl {
    type string;
    description
      "An absolute path to a CRL for the issuer used by the Vendor to sign Owner Certificates. The CRL should be as up to date as possible. This leaf is optional as it is primarily to support deployments where the device is unable to download the CRL from the CRL distribution point URLs listed in the Vendor’s trust anchor certificate."
  }
}
point URLs listed in the Vendor’s trust anchor
certificate."
}
}

container boot-image {
  presence
  "Only present when boot image information has been configured";
  description
  "It is intended that the device will fetch this container
  as a whole, as it contains values that need to be
  processed together.";
  leaf name {
    type string;
    mandatory true;
    description
    "The name of the image of software the device is expected
    to be running.";
  }
  leaf path {
    type string;
    mandatory true;
    description
    "An absolute path to the boot-image file hosted on this
    Bootstrap server.";
  }
  leaf signature {
    type string;
    mandatory true;
    description
    "The signature over the concatenation of the previous two
    leafs using the organization’s private key.";
  }
}

container configuration {
  description
  "It is intended that the device will fetch this container
  as a whole, as its contents need to be processed together.";
  anyxml config {
    mandatory true;
    description
    "Any configuration data model known to the device. It may
    contain Vendor-specific and/or standards-based data models.
    An example configuration using a couple IETF-defined data
    models is presented the Appendix of RFC XXXX.";
  }
  leaf signature {

type string;
  mandatory true;
  description
   "The signature over the config leaf using the
    organization’s private key."
  
}
}

rpc notification {
  input {
    leaf unique-id {
      type string;
      mandatory true;
    }
    leaf type {
      type enumeration {
        enum boot-image-missing {
          description
           "Indicates that the device got an error when trying to
            access the provided URL";
        }
        enum boot-image-invalid {
          description
           "Indicates that the device had a problem processing the
            boot-image file (corruption)";
        }
        enum image-name-mismatch {
          description
           "Indicates that the processed boot-image contains a name
            other than provided";
        }
        enum voucher-invalid {
          description
           "Indicates that the device had a problem processing the
            voucher (chain verification failed, revoked crl)";
        }
        enum owner-cert-invalid {
          description
           "Indicates that the device had a problem processing the
            voucher (chain verification failed, revoked crl)";
        }
        enum owner-id-mismatch {
          description
           "Indicates that the owner-id in the voucher does not
            match the one inside the owner-cert";
        }
      }
    }
  }
}
enum signature-invalid {
  description
  "Indicates that the signature could not be verified using the owner-cert";
}

enum bootstrap-complete {
  description
  "Indicates that the device successfully processed the bootstrap data. At this point, the device is running the required boot-image and configuration. A device is expected to only send this notification once, assuming it does not receive an error in the HTTP response from the Bootstrap Server.";
}

mandatory true;

leaf message {
  type string;
  description
  "A human-readable value that might provide useful information";
}

4. Device

Devices supporting ZeroTouch MUST have the preconfigured factory default state and bootstrapping logic described in the following sections.

4.1. Factory Default State
1. Devices MUST be manufactured with a list of default Bootstrap Servers. Each Bootstrap Server may be identified via a hostname or an IP address. This may be an empty list if for some reason the Vendor prefers to force its devices to have to discover Bootstrap Servers from a DHCP server.

2. Devices MUST be manufactured with a list of trust anchor certificates that can be used to authenticate Bootstrap Server connections with. To support Bootstrap Servers discovered from a DHCP server, these certificates SHOULD include public certificate authorities, such as those that are included in a web browser.

3. Devices MUST be manufactured with the trust anchor certificate for Owner certificates that the Vendors provide to business organizations when they enroll in the Vendor’s ZeroTouch program. This trust anchor certificate is later used by the device to validate the Owner certificate it downloads from the Bootstrap Server.

4. Devices MUST be manufactured with the trust anchor certificate for the device ownership vouchers that the Vendors provide to organizations when it ships out an order of ZeroTouch devices. This trust anchor certificate is later used by the device to validate the Ownership Vouchers it downloads from the Bootstrap Server.
5. 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 a globally unique device identifier (e.g., serial number). The device MUST also possess any intermediate certificates between the IDevID certificate and the Vendor’s IDevID trust anchor certificate. These certificates are later used by the device to identify itself when it calls home. In particular, these certificates are to be used by the device’s NETCONF server, either as its SSH host-key or its TLS server certificate. Please see NETCONF Call Home [ draft-ietf-netconf-call-home ] for more information.

6. Device MUST be manufactured with a private key that corresponds to the public key encoded in its IDevID certificate. This private key SHOULD be securely stored, ideally by a cryptographic processor (e.g., a TPM).

4.2. Boot Sequence

```
Power On
    v No
1. Running default config? --------> Boot normally
      v Yes
      v No
2. Able to reach DHCP server? --------> Boot normally
      v Yes
      v
3. Prepend any additional Bootstrap Servers discovered
      v No
4. Able to bootstrap off any Bootstrap Server? --------> Boot normally
      (see next diagram for drill-down details)
      v Yes
      v
5. Run with new configuration
```

These interactions are described next.

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 tries to obtain a dynamic network assignment from a DHCP server. If it is unable to reach a DHCP server, it boots normally.

3. If the DHCP server’s offer includes the ZeroTouch Information DHCP option defined in Section 7.1, the device prepends the specified Bootstrap Servers to its factory default list.

4. The device iterates over its list of Bootstrap Servers, as described in the next section. If it is unable to bootstrap itself off any of the servers, it boots normally.

5. If the device was able to bootstrap itself off any of the Bootstrap Servers, it runs with the new configuration merged into its running datastore.

Following are the actions performed by the device when bootstrap off a Bootstrap Server (step #4 the in previous diagram).
Connect to port 443
| v | No
1. Able to validate server certificate? ———> Exit
    | v | Yes  
    | v | No
2. Able to validate ownership voucher? ———> Post notification and exit
    | v | Yes  
    | v | No
3. Able to validate owner certificate? ———> Post notification and exit
    | v | Yes  
    | v | No
4. Able to validate boot image info? ———> Post notification and exit
    | v | Yes  
    | v | No
5. Need to install boot image? ———> Install and reboot
    | v | Yes  
    | v | No
6. Able to validate configuration? ———> Post notification and exit
    | v | Yes  
    | v
7. Merge configuration into running datastore
    | v
8. Post bootstrap complete notification and exit

These interactions are described next.

1. As part of the HTTPS connection, the device will need to authenticate the server certificate presented by the Bootstrap Server. The device authenticates the server certificate using path-validation to one of its preconfigured Bootstrap Server trust anchors. If the device is unable to authenticate the server’s certificate, it abandons this Bootstrap Server and exits.

2. The device downloads the ownership voucher from the Bootstrap Server. The device validates the voucher is signed by its Vendor, using its preconfigured trust anchor for device ownership vouchers. The device also validates that its unique identifier is listed by the voucher. If the device is unable to validate the voucher or can not find its unique identifier listed, it
posts a notification message to that effect and abandons this Bootstrap Server.

3. The device downloads the owner certificate from the Bootstrap Server. The device validates that this certificate is signed by its Vendor, using path-validation to its preconfigured trust anchor for owner certificates. The device also validates that the organization identifier is the same as listed in the ownership voucher, validated in step #2. If the device is unable to validate the certificate or the owner identifier does not match, it posts a notification message to that effect and abandons this Bootstrap Server.

4. The device tries to download the boot image information. If no boot image information is available, it skips the remainder of this step. Otherwise, the device validates the boot image information using the public key from the owner certificate obtained in step #3. If it is unable to authenticate the boot image information, it posts a notification message to that effect and abandons this Bootstrap Server.

5. The device checks if the specified boot-image name matches what the device is currently running. If there is a mismatch, device downloads the new image from the Bootstrap Server and installs it. It is expected that the device will reboot itself in order to activate the new image and, further, that doing so preserves its factory default state such that it will return to this same check again, but then running the correct image. If the device is unable to install the boot-image, it posts a notification message to that effect and abandons this Bootstrap Server.

6. The device downloads the configuration from the Bootstrap Server and validates the configuration using the public key from the owner certificate obtained in step #3. If it is unable to authenticate the configuration, it posts a notification message to that effect and abandons this Bootstrap Server.

7. The device merges the configuration into its running datastore. It is expected that this configuration will provide the information necessary for the device to establish a secure NETCONF connection to its NMS using NETCONF Call Home ([draft-ietf-netconf-call-home]).

8. The device posts a bootstrap completion notification message to the Bootstrap Server and exits.
5. Network Management System (NMS)

5.1. Overview

It is expected that the bootstrapping configuration will guide the device to initiate a secure NETCONF connection to the NMS using NETCONF Call Home [draft-ietf-netconf-call-home]. This section describes what the NMS needs to do to ensure security for the device's connection.

5.2. Precondition

1. In order to authenticate a device, the NMS MUST possess the IDevID trust anchor provided by its Vendor to enable verification of the device's IDevID certificate. Specifically, the NMS uses this certificate to validate the identity certificate a device presents when negotiating the SSH or TLS transport for the NETCONF Call Home connection [draft-ietf-netconf-call-home]. Because an NMS may interoperate with multiple vendors, and a vendor may have more than one trust anchor for signing its devices IDevID certificates, this generalizes into the NMS needing a list of trust anchor certificates. These certificates SHOULD be stored in a way that prevents tampering, which is why they are shown in read-only storage in the diagram.
2. In order for the NMS to validate that a specific device connecting to it is legitimate, it MUST have a list of expected unique device identifiers (e.g., serial-numbers). The unique identifier encoded into the device’s IDevID certificate MUST match one of the expected identifiers in order for a device to be considered legitimate.

3. The NMS must have login credentials for each device. These credentials may be, for instance, a private key used for SSH or TLS client authentication. It is expected that a device is able to authenticate the NMS's credentials by virtue of the configuration it downloads from the Bootstrap Server. These private-keys SHOULD be stored securely, such that they can not be easily compromised.

5.3. Connection Handling

When receiving a NETCONF call home connection from a device, the NSM completes the connection as specified NETCONF Call Home [draft-ietf-netconf-call-home].

6. Security Considerations

6.1. 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 a notAfter 99991231235959Z). Given the long-lived nature of these certificates, it is paramount to use a strong key length (e.g., 512-bit ECC). Vendors SHOULD deploy Online Certificate State Protocol (OCSP) responders or CRL Distribution Points (CDP) to revoke certificates in case necessary.

6.2. 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.
7. IANA Considerations

7.1. ZeroTouch Information DHCP Options

The following registrations are in accordance to RFC 2939 for "BOOTP Vendor Extensions and DHCP Options" registry maintained at http://www.iana.org/assignments/bootp-dhcp-parameters.

7.1.1. DHCP v4 Option

Tag: XXX

Name: Zero Touch Information

Description: Returns a list of null-terminated Configuration Server hostnames and/or IP addresses.

| XXX | n  | svr1 | svr2 | ...
|-----|-----|------|------|----|

Reference: RFC XXXX

7.1.2. DHCP v6 Option

Tag: YYY

Name: Zero Touch Information

Description: Returns a list of null-terminated Configuration Server hostnames and/or IP addresses.

| YYY | n  | svr1 | svr2 | ...
|-----|-----|------|------|----|

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): David Harrington, Dean Bogdanovic, Martin Bjorklund, Stephen Hanna, Wes Hardaker, Russ Mundy, Reinaldo Penno, Randy Presuhn, 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.

9. Normative References


Appendix A. Examples

A.1. Ownership Voucher

Following describes an example data-model for an Ownership Voucher. Real vouchers are expected to be encoded in a Vendor-specific format outside the of scope for this draft.

A tree diagram describing an Ownership Voucher:

module: ietf-zerotouch-ownership-voucher
  +--rw voucher
    +--rw unique-id* string
    +--rw owner-id string
    +--rw created-on yang:date-and-time
    +--rw expires-on? yang:date-and-time
    +--rw signature string

The YANG module for this example voucher:

<CODE BEGINS> file "ietf-zerotouch-ownership-voucher@2015-03-09.yang"

module ietf-zerotouch-ownership-voucher {

  prefix "ztov";

  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>
      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 the format for a ZeroTouch ownership voucher, which is produced by Vendors, relayed by Bootstrap Servers, and consumed by devices. The purpose of the voucher is to enable a device to ascertain the identity of its rightful owner, as"
certified by its Vendor.

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 "2015-03-09" {
  description
    "Initial version";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF Call Home";
}

// top-level container
container voucher {
  leaf-list unique-id {
    type string;
    min-elements 1;
    description
      "The unique identifier (e.g., serial-number) for a device. The value
      must match the value in the device’s IDevID certificate. A device
      uses this value to determine if the voucher applies to it."
  }
  leaf owner-id {
    type string;
    mandatory true;
    description
      "A Vendor-assigned value for the rightful owner of the devices
      enumerated by this voucher. The owner-id value must match the
      value in the owner-certificate below"
  }
  leaf created-on {
    type yang:date-and-time;
    mandatory true;
    description
      "The date this voucher was created"
  }
  leaf expires-on {

A.2. Bootstrap Server’s API

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:
devices/device=123456/ownership-voucher

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:
devices/device=123456/owner-certificate

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:
devices/device=123456/boot-image

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:
devices/device=123456/configuration

POST https://example.com/restconf/operations/ietf-zerotouch-bootstrap-
server:notification

A.3. Bootstrap Configuration

This example illustrates a configuration enabling secure NETCONF
call-home using standards-based YANG modules.
<?xml version="1.0"?>
<configuration>
  <!-- from ietf-system.yang -->
  <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
    <authentication>
      <user>
        <name>admin</name>
        <ssh-key>
          <name>admin’s rsa ssh host-key</name>
          <algorithm>ssh-rsa</algorithm>
          <key-data>AAAAB3NzaC1yc2EAAAADQABAAAAABAgQDeJMV8zrtsi8CgEsRCjCzfve2m6zD3awSBPrh7ICggLQvHVbFL89eHLuecStKL3HrEgXaI/O2MwjE1lG9YxLzeS5p2ngzK61vikiU5qfMukeBohFTrDZ0bUtrF+HMLlTRncCVCcWAiw1o9IDGDAuw6G45gLcHalHMMbtQxKn2dzU9kxf/f32S5G76fy6sA5vg75lq0PFjXXft2CAhIn8xwYRZy6r/2N9PMJ2Dnpvq4H2DKqBIe340jWqEJvA7lVeJYq14unq4ilog/+CiumTkmQIWRgIo4FCzYk09nVRE6fOSLLf6gakMNQZgQ892uWjCW1Glqn2mPibp2Go1</key-data>
        </ssh-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>
Appendix B. Change Log

B.1. ID to 00

- Major structural update; the essence is the same. Most every section was rewritten to some degree.
- Added a Use Cases section
- Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram
- Removed support for physical presence or any ability for Configlets to not be signed.
- Defined the ZeroTouch Information DHCP option
- Added an ability for devices to also download images from Configuration Servers
- Added an ability for Configlets to be encrypted
- Now Configuration Servers only have to support HTTP/S - no other schemes possible

B.2. 00 to 01

- Added boot-image and validate-owner annotations to the "Actors and Roles" diagram.
- Fixed 2nd paragraph in section 7.1 to reflect current use of anyxml.
- Added encrypted and signed-encrypted examples
- Replaced YANG module with XSD schema
- Added IANA request for the ZeroTouch Information DHCP Option
- Added IANA request for media types for boot-image and configuration

B.3. 01 to 02

- Replaced the need for a Configuration Signer with the ability for each NMS to be able to sign its own configurations, using Vendor signed Ownership Vouchers and Owner certificates.
Renamed Configuration Server to Bootstrap Server, a more representative name given the information devices download from it.

Replaced the concept of a Configlet by defining a southbound interface for the Bootstrap Server using YANG.

Removed the IANA request for the boot-image and configuration media types

Authors’ Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

Joe Clarke
Cisco Systems
EMail: jclarke@cisco.com

Mikael Abrahamsson
T-Systems
EMail: "mikael.abrahamsson@t-systems.se"
Abstract

This document discusses several scenarios that multiple replies for a single request are needed, with the ability to terminate the replies at any time. Such scenarios are not well supported by current NETCONF (Network Configuration) protocol. Several solution candidates are also discussed in this document.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

The message procedures of NETCONF [RFC6241] are based on RPC (Remote Procedure Call) interactions. A NETCONF client/server sends a <rpc> message to the counterpart and then receives a replying <rpc-reply> message.

In some situations, it might need multiple <rpc-reply> messages for a <rpc> request. For example, the the <rpc-reply> message might be very large that it needs to be fragmented into multiple small ones; or some operations (e.g. ping) need persistent replies till such time that a terminating condition is encountered or when the operation is cancelled.

This document discusses such kind of multiple replies scenarios, analyzes the issues of current NETCONF protocol in supporting these scenarios, and proposes several candidate solutions for discussion.
2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Terminology:

CLI: Command Line interface

DOM: Document Object Model, which is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects.

RPC: Remote Procedure Call

SAX: Simple API for XML, which is an event sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the DOM. Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

libxml: a software library for parsing XML documents.

<get-block>: a capability and operation defined in this document to handle large size <rpe-reply> messages.

3. Scenarios and Problems

This section discusses several scenarios where multiple replies might be needed, and analyzes the problems of current NETCONF protocol in supporting these scenarios.

3.1. Bulk <rpc-reply>

The <rpc-reply> message might be very large in following situations:

- retrieving a large amount of routes in a core router
- retrieving a large interface statistics list
- doing a full-synchronizing with a device
Current we already have some methods of processing bulk replies as the following. However, there are some issues as analyzed below.

1) Stream-Oriented Handling

Stream-Oriented handling mainly includes the following two aspects:

- The server encapsulates the large size replying data in a `<rpc-reply>` message and streams it to the client through TCP protocol.
- The client parses the received `<rpc-reply>` content in a stream-oriented way. More specifically, the client could utilize SAX parsing to instantly parse the received content without waiting for the whole message been transported.

The problems are:

- Stream-Oriented method lacks the capability of discontinuing large size processing in the server. It would cause unnecessary resource/performance cost in the devices if the NMS has already got the intended portion or just canceled by the administrators.
- Another problem is the implementation of SAX parsing is more complex than DOM parsing in the Netconf client. More computing burden will be taken in Netconf client to support SAX parsing.

2) Requesting a Portion of Data

The clients actively limit the search range of the data so that the servers only need to reply with a part of the large size data. Thus the clients could control the replies in a reasonable size. One example is that the clients get a list of the content, and provide a start offset and a max-count, to get a portion at a time.

The problems are:

- This method has an implication that the client needs to know the list/index of the intended large size data in advance before it starting the search request. It can’t fit the scenarios of real-time on-demand data retrieving. And there is no standard to specify the list/index format in a uniform way. Thus it is only suitable for private implementation, thus multi-vendor interaction is not supported.
- More important, it is just an indirect way to solve the problem. It could not fit the scenarios where the client just needs the whole large size data in the server.
3.2. Persistent <rpc-reply>

One of the operations that CLI offers today is the ability to issue an operation that might result in multiple responses being returned, till such time that a terminating condition is encountered or when the operation is cancelled. An example of such an operation is when the ping or a traceroute command is issued. In the former case, the operation can continue sending responses back till it is cancelled, while in the latter case there is usually a terminating condition that stops the responses.

NETCONF protocol as defined today sends a single Remote Procedure Call (RPC) request and expects a single reply to that request. The "persistent" operation defined above expects multiple responses for a single request, till such time a terminating condition is encountered.

3.3. Long time <rpc-reply>

Some operations might take a long time to perform (e.g. network link performance validation), so there could be multiple responses for the request. For example, initial responses returns handle which the client uses to monitor progress till the final result. The client should be able to cancel the request at any time.

4. Requirements for NETCONF

Given above mentioned multiple-replies scenarios and problems, the requirements for NETCONF protocol could be summarized as a mechanism with the following abilities:

- be able to generate and handle multiple <rpc-reply> messages for a given <rpc> message
- be able to terminate the <rpc-reply> at any time
- be able to cancel the request in pipeline scenarios

5. Candidate Solutions

(Editor notes: this section discusses several possible solutions. The fragmentation mechanism is the original proposal of the draft. The other two were proposed during mailing list discussion by Andy Bierman and Juergen Schoenwaelder respectively. We include all of them for discussion and solution selection.)
5.1. Fragmentation Mechanism

This section proposes a method of extending the NETCONF protocol to allow handling large size data as fragmented <rpc-reply> messages. The fragmentation is done at the NETCONF level, so it allows the NETCONF client to terminate the large size data processing momentarily by protocol interactions; and also allows the fragmented messages to be instantly parsed piece by piece. Specifically, the fragmentation is achieved through a newly defined <get-block> capability and relevant operations.

5.1.1. Design Requirements

Two essential requirements of the fragmentation mechanism are:

- It needs to allow the NETCONF client to terminate the large size data processing momentarily by protocol interactions. In the proposed mechanisms in this draft, when the NETCONF server replies the client an <rpc-reply> fragmentation, it will wait the response from the client that whether it needs to send the next fragmentation. So if the initiator has got the intended portion, it could terminate the large size process immediately.

- It needs to allow the NETCONF client to instantly parse the fragmentations piece by piece through the more widely supported DOM parsing. So in this document, it specifies that each <rpc-reply> fragmentation MUST be in a complete XML form.

5.1.2. <get-block> extension

- Function

  The devices can only use <get-block> operation when the Get-block capability was announced.

  The <get-block> fragmentation rules are:

  A. There should be a Max-Size for fragmentation. [Open Question] Should there be a clear specification of the size? E.g. 64K bytes.

  B. When the message reaches the Max-Size, it is sent to the client and the next message could be created in advance.

  C. Different records from one same table could be put into different <rpc-reply> messages
D. All of the fields in one record MUST be put into one <rpc-reply> message.

E. XML syntax MUST be complete in each fragmented message, so that each fragmentation could be parsed individually.

F. If the record(s) of the child node(s)/table(s) and the parent node(s)/table(s) are replied in different fragmentations, the child node/table fragmentations MUST include the path and index information of all the ancestor node(s)/table(s) in a hierarchical mode.

- Parameters

<discard/>: in <get-block> operation, if the <discard/> parameter is conveyed, it means the operation is terminated. Then it doesn’t need to reply the remaining fragmentations.

- Successful Operation Reply

A <rpc-reply> message conveying a <data> element indicates the operation is successful.

If there exists a next fragment, then an set-id attribute MUST be included in the <rpc-reply> message. The attribute set-id is used to identify different fragment sets.

- Exception Handling

After the NETCONF server replies a fragment, if there is no corresponding Get-block request from the client in a reasonable period (the time valued to be specified in the future), then the server release the offset of the replying data and cannot use <get-block> operation anymore, and the remaining data needs to be replied.

Please refer to Appendix A.1 for an example.

5.2. Linked Replies

Another solution is to change or augment NETCONF at some point in time such that an <rpc> can lead to a sequence of <rpc-reply> with a suitable cancel mechanism. A simple approach is to add a linked-replies capability. If a server announces "linked-replies" capability and the client supports it as well, the client can add an additional parameter to an rpc to indicate the possible use of linked-replies.
Please refer to Appendix A.2 for an example.

This would address the concern of large data retrievals but would also allow long running asynchronous rpcs (the ping or traceroute example). This approach may lead to better support for asynchronous rpcs and rpcs that potentially return very large chunks of data than trying to solve this problem without enhancements of the rpc layer. Design details concerning data merging, error handling, how to send a cancel for a given link-id (e.g., by sending a new <rpc-cancel> message with a matching link-id) and whether it is necessary to negotiate linked rpc-reply sizes or whether it is good enough for the server to decide freely as it likes etc. need further study.

5.3. Subtree Iteration

An "iterator" approach allows a list resource to be retrieved in chunks. An RPC function could be added to do the iteration operation.

Please refer to Appendix A.3 for an example.

There is a problem with rapidly changing lists (could get repeat entries on miss some entries).

(Editor’s note: this solution is specifically for bulk response problem. It doesn’t support persistent response and long time response since it follows one <rpc> and one <rpc-reply> approach.)

6. Security Considerations

TBD.

7. IANA Considerations

This draft does not request any IANA action.

8. Acknowledgements

Gang Yan and Shouchuan Yang made significant contribution to form the draft. Valuable comments were received from Andy Bierman, Juergen Schoenwaelder, Balazs Lengyel, Martin Bjorklund, Chong Feng and some other people in Netconf working group.

This document was produced using the xml2rfc tool [RFC2629].
(initialllly prepared using 2-Word-v2.0.template.dot. )
9. Normative References


Appendix A. Examples of the Candidate Solutions

A.1. <get-next> (RPC Fragmentation) Example

Example 1: Get the next fragment
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
 <get-config>
 <source>
 <running/>
 </source>
 <filter type="subtree">
 <top xmlns="http://example.com/schema/1.2/config">
 <users/>
 </top>
 </filter>
 </get-config>
</rpc>

<rpc-reply message-id="101"
 xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
 xmlns:hw=http://example.com/NETCONF/capability/base/1.0
 hw:set-id="101">
 <data>
 <top xmlns="http://example.com/schema/1.2/config">
 <users>
 <user>
 <name>root</name>
 <type>superuser</type>
 <full-name>Charlie Root</full-name>
 <company-info>
 <dept>1</dept>
 <id>1</id>
 </company-info>
 </user>
 <!-- additional <user> elements appear here... -->
Example 2: Abandon the remaining fragments
<rpc message-id="103"
     xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <get-block xmlns=http://example.com/NETCONF/capability/base/1.0
             set-id="1">
    <discard/>
  </get-block>
</rpc>

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

Example 3: Following is an example of the rule f in Section 4.1.2.
The child eTable is in a different message with the parents aTable->bTable->cTable->dTable. Then the path and index information of all the ancestors MUST included in the search data.

```xml
<aTable>
  <aEntity>
    <aIndex1>
    </aEntity>
  <bTable>
    <bEntity>
      <bIndex1>
      </bEntity>
    </eTable>
    <eEntity>
      <eIndex1>
        <ef2>
          <ef3>
        </ef2>
      </eIndex1>
    </eTable>
    </bTable>
  </aTable>
</aTable>
```

A.2. Linked-replies Example

Here is what a new client might do if it wants to use linked replies:

```xml
<rpc message-id="101" link-id="123" xmlns="..."/>
</rpc>
```

The server can either simply send an rpc-reply or it starts sending linked replies, e.g.:

```xml
<rpc-reply message-id="101" next-message-id="101" link-id="123" xmlns="..."/>
</rpc-reply>

<rpc-reply message-id="101" next-message-id="101" link-id="124" xmlns="..."/>
</rpc-reply>

<rpc-reply message-id="101" link-id="125" xmlns="..."/>
</rpc-reply>
```

A.3. `<get-list>` (Subtree Iteration) Example

```xml
rpc get-list {
  input {
    leaf target {
      type schema-instance-identifier;
    }
  }
}
```
description "Identifies subtree to retrieve."
}
leaf start {
    type uint32;
    default 0;
    description "Number of entries to skip before starting retrieval"
}
leaf max-entries {
    type uint32 { range "1..max"; }
    default 25;
    description "Maximum number of list entries to retrieve"
}
}
output {
    anyxml data {
        description "Contains the requested data"
    }
}

<rpc>
    <get-list>
        <target>/if:interfaces/if:interface</target>
    </get-list>
</rpc>

<rpc-reply>
    <data>
        <interfaces>
            <interface> .... first entry </interface>
            ...
            <interface> .... 25th entry </interface>
        </interfaces>
    </data>
</rpc-reply>

<rpc>
    <get-list>
        <target>/if:interfaces/if:interface</target>
        <start>25</start>
    </get-list>
</rpc>

<rpc-reply>
<data>
  <interfaces>
    <interface> .... 26th entry </interface>
    ...
    <interface> .... 50th entry </interface>
  </interfaces>
</data>
</rpc-reply>

Authors’ Addresses

Bing Liu
Huawei Technologies
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: leo.liubing@huawei.com

Guangying Zheng
Huawei Technologies
Huawei Nanjing R&D Center
101 Software Avenue, Yuhua District, Nanjing, Jiangsu, 210012
P.R. China

Email: zhengguangying@huawei.com

Mahesh Jethanandani
Ciena Corporation
3939 North 1st Street
San Jose, CA  95134
USA

Phone: 408.904.2160
Fax: 408.436.5582
Email: mjethanandani@gmail.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net
Abstract

This document defines a capability-based extension to the Network Configuration Protocol (NETCONF) that allows time-triggered configuration and management operations. This extension allows NETCONF clients to invoke configuration updates according to scheduled times, and allows NETCONF servers to attach timestamps to the data they send to NETCONF clients.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 26, 2014.

Copyright Notice

Copyright (c) 2014 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
1. Introduction

The Network Configuration Protocol (NETCONF) defined in [RFC6241] provides mechanisms to install, manipulate, and delete the
configuration of network devices. NETCONF allows clients to configure and monitor NETCONF servers using remote procedure calls (RPC).

NETCONF, as defined in [RFC6241], is asynchronous; when a client invokes an RPC, it has no control over the time at which the RPC is executed, nor does it have any feedback from the server about the execution time.

Time-based configuration ([HotSDN], [TimeTR]) can be a useful tool that enables an entire class of coordinated and scheduled configuration procedures. Time-triggered configuration allows coordinated network updates in multiple devices; a client can invoke a coordinated configuration change by sending RPCs to multiple servers with the same scheduled execution time. A client can also invoke a time-based sequence of updates by sending n RPCs with n different update times, T1, T2, ..., Tn, determining the order in which the RPCs are executed.

This memo defines the time capability in NETCONF. This extension allows clients to determine the scheduled execution time of RPCs they send. It also allows a server that receives an RPC to report its actual execution time to the client.

2. Conventions used in this document

2.1. Keywords

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 [RFC2199].

2.2. Abbreviations

NETCONF Network Configuration Protocol
RPC Remote Procedure Call

2.3. Terminology

- Capability [RFC6142]: A functionality that supplements the base NETCONF specification.
- Client [RFC6142]: Invokes protocol operations on a server. In addition, a client can subscribe to receive notifications from a server.
3. Using Time in NETCONF

3.1. The Time Capability in a Nutshell

The :time capability provides two main functions:

- **Scheduling:** When a client sends an RPC to a server, the RPC message MAY include a scheduled time, Ts (see Figure 1). The server then executes the RPC at the scheduled time Ts, and once completed the server can respond with an RPC reply message.

- **Reporting:** When a client sends an RPC to a server, the RPC message MAY include a get-time element (see Figure 2), requesting the server to return the execution time of the RPC. In this case, after the server performs the RPC it responds with an RPC reply that includes the execution time, Te.

```
RPC _________      \        \        \        \        \        \    ----> time
executed \       \       \       \       \       \ server  ---------------+-------------        ---->
          \       \                \ rpc-reply
Ts        \       \                \ (Ts)/ \         /                  /
client  -----------------------------        
```

Figure 1 Scheduled RPC
The two scenarios discussed above imply that a third scenario can also be supported (Figure 3), where the client invokes an RPC that includes a scheduled time, Ts, as well as the get-time element. This allows the client to receive feedback about the actual execution time, Te. Ideally, Ts=Te. However, the server may execute the RPC at a slightly different time than Ts, for example if the server is tied up with other tasks at Ts.

3.2. Notifications and Cancellation Messages

Notifications

As illustrated in Figure 1, after a scheduled RPC is executed the server sends an rpc-reply. The rpc-reply may arrive a long period of time after the RPC was sent by the client, leaving the client without a clear indication of whether the RPC was received.
This document defines a new notification, the netconf-scheduled-message notification, which provides an immediate acknowledgement of the scheduled RPC.

The netconf-scheduled-message is sent to the client if it is subscribed to the NETCONF notifications [RFC6470]; as illustrated in Figure 4, when the server receives a scheduled RPC it sends a notification that includes the message-id of the scheduled RPC.

```
RPC ____________
executed \   
\/
Ts
server -----------------+---------        ----> time
\  
rpc /    
\notifi-
(Ts)/    
\rpc-reply
\     
\cation
/  
\/
client -----------------------------
```

Figure 4 Scheduled RPC with Notification

Cancellation Messages

A client can cancel a scheduled RPC by sending a <cancel-schedule> RPC.

The <cancel-schedule> RPC, defined in this document, can be used to perform a coordinated all-or-none procedure, where either all the servers perform the operation on schedule, or the operation is aborted.

Example. The client sends scheduled RPC messages to server 1 and server 2, both scheduled to Ts. Server 1 sends a notification that indicates it has successfully scheduled the RPC, while server 2 replies with an unknown-element error [RFC6241] that indicates that it does not support the time capability. The client sends a <cancel-schedule> RPC to server 1, and receives an rpc-reply. The message exchange between the client and server 1 in this example is illustrated in Figure 5.
3.3. Synchronization Aspects

The time capability defined in this document requires clients and servers to maintain clocks. It is assumed that clocks are synchronized by a method that is outside the scope of this document, e.g., [NTP] or [IEEE1588].

This document does not define any requirements pertaining to the degree of accuracy of performing scheduled RPCs. Note that two factors affect how accurately the server can perform a scheduled RPC; one factor is the accuracy of the clock synchronization method used to synchronize the clients and servers, and the second factor is the server’s ability to execute real-time configuration changes, which greatly depends on how it is implemented. Typical networking devices are implemented by a combination of hardware and software. While the execution time of a hardware module can typically be predicted with a high level of accuracy, the execution time of a software module may be variable and hard to predict. A configuration update would typically require the server’s software to be involved, thus affecting how accurately the RPC can be scheduled.

Another important aspect of synchronization, is monitoring; a client should be able to check whether a server is synchronized to a reference time source. Typical synchronization protocols, such as the Network Time Protocol ([NTP], [RFC5907]) provide the means to verify that a clock is synchronized to a time reference by querying its Management Information Base (MIB). The get-time feature defined in this document (see Figure 2) allows a client to obtain a rough estimate of the time offset between the client’s clock and the server’s clock.
Since servers do not perform configuration changes instantaneously, the processing time of an RPC should not be overlooked. The scheduled time always refers to the start time of the RPC, and the execution time always refers to its completion time.

3.4. Scheduled Time Format

The scheduled time and execution time fields in RPC messages use a common time format field.

The time format used in this document is the date-and-time format, that is defined in Section 5.6 of [RFC3339] and in Section 3 of [RFC6021].

```yang
leaf scheduled-time {
  description
  "The time at which the RPC is scheduled to be performed.";
  type yang:date-and-time;
}

leaf execution-time {
  description
  "The time at which the RPC was executed.";
  type yang:date-and-time;
}
```

3.5. Scheduling Tolerance

When a client sends an RPC that is scheduled to Ts, the server MUST verify that the value Ts is not too far in the past or in the future. As illustrated in Figure 6, the server verifies that Ts is within the scheduling tolerance range.
The scheduling tolerance is determined by two parameters, sched-max-future and sched-max-past. These two parameters use the time-interval format (Section 3.7.), and their default value is 15 seconds.

If the scheduled time, Ts is within the scheduling tolerance range, the scheduled RPC is performed; if Ts occurs in the past and within the scheduling tolerance, the server performs the RPC as soon as possible, whereas if Ts is a future time, the server performs the RPC at Ts.

If Ts is not within the scheduling tolerance range, the server responds with an error message [RPC6241] with a bad-element error-tag. An example is provided in Section 5.3.

3.6. Near Future Scheduling vs. Far Future Scheduling

The scheduling bound defined by sched-max-future guarantees that every scheduled RPCs is restricted to a near future scheduling time.

The scheduling mechanism defined in this document is intended for near future scheduling, on the order of seconds. Far future scheduling is outside the scope of this document.

The challenge in far future scheduling is that during the long period between the time at which the RPC is sent and the time at which it is scheduled to be executed various external events may occur, e.g., the client may fail or reboot, or the client access permissions may be changed. In these cases if the server performs the scheduled operation it may perform an action that is inconsistent with the
current network policy, or inconsistent with the currently active clients.

Near future scheduling guarantees that external events such as the examples above have a low probability of occurring during the sched-max-future period, and even when they do, the period of inconsistency is limited to sched-max-future, which is a short period of time.

3.7. Time Interval Format

The time-interval format is used for representing the length of a time interval, and is based on the date-and-time format. It is used for representing the scheduling tolerance parameters, as described in the previous section.

While the date-and-time type uniquely represents a specific point in time, the time-interval type defined below can be used to represent the length of a time interval without specifying a specific date.

The time-interval type is defined as follows:

```c
typedef time-interval {
    type string {
        pattern '\d{2}:\d{2}:\d{2}(\.|\d+)?';
    }
}
```

4. Time Capability

The structure of this section is as defined in Appendix D of [RFC6241].

4.1. Overview

A server that supports the time capability can perform time-triggered operations as defined in this document.

A server implementing the :time capability:

- MUST support the ability to receive <rpc> messages that include a time element, and perform a time-triggered operation accordingly.

- MUST support the ability to include a time element in the <rpc-reply> messages that it transmits.
4.2. Dependencies

With-defaults Capability

The time capability YANG module (Appendix A.) uses default values, and thus it is assumed that the with-defaults capability [RFC6243] is supported.

4.3. Capability Identifier

The :time capability is identified by the following capability string (to be assigned by IANA - see Section 7.):

urn:ietf:params:netconf:capability:time:1.0

4.4. New Operations

<cancel-schedule>

The cancel-schedule RPC is used for cancelling an RPC that was previously scheduled.

A cancel-schedule RPC MUST include the <cancelled-message-id> element, which specifies the message ID of the scheduled RPC that needs to be cancelled.

A cancel-schedule RPC MAY include the <get-time> element. In this case the rpc-reply includes the <execution-time> element, specifying the time at which the scheduled RPC was cancelled.

4.5. Modifications to Existing Operations

Three new elements are added to all existing operations:

- <scheduled-time>
  
  This element is added to the input of each operation, indicating the time at which the server is scheduled to invoke the operation. Every <rpc> message MAY include the <scheduled-time> element. A server that supports the :time capability and receives an <rpc> message with a <scheduled-time> element MUST perform the operation as close as possible to the scheduled time.

  The scheduled-time element uses the date-and-time format (Section 3.4.).
o <get-time>
   This element is added to the input of each operation. An <rpc>
   message MAY include a <get-time> element, indicating that the
   server MUST include an <execution-time> in its corresponding <rpc-
   reply>.

o <execution-time>
   This element is added to the output of each operation, indicating
   the time at which the server completed the operation. An <rpc-
   reply> MAY include the <execution-time> element. A server that
   supports the :time capability and receives an operation with the
   <get-time> element MUST include the execution time in its
   response.

   The execution-time element uses the date-and-time format
   (Section 3.4.).

4.6. Interactions with Other Capabilities

Confirmed Commit Capability

The confirmed commit capability is defined in Section 8.4 of
[RFC6241]. According to [RFC6241], a confirmed <commit> operation
MUST be reverted if a confirming commit is not issued within the
timeout period (which by default is 600 seconds).

When the time capability is supported, and a confirmed <commit>
operation is used with the <scheduled-time> element, the confirmation
timeout MUST be counted from the scheduled time, i.e., the client
begins the timeout measurement starting at the scheduled time.

5. Examples

5.1. <scheduled-time> Example

   The following example extends the example presented in Section 7.2 of
   [RFC6241] by adding the time capability. In this example, the
   <scheduled-time> element is used to specify the scheduled execution
time of the configuration update (as shown in Figure 1).

   <rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
      <target>
         <running/>
      </target>
   </edit-config>
</rpc>
<scheduled-time
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-time">
  2015-10-21T04:29:00.235Z
</scheduled-time>
<config>
  <top xmlns="http://example.com/schema/1.2/config">
    <interface>
      <name>Ethernet0/0</name>
      <mtu>1500</mtu>
    </interface>
  </top>
</config>
</rpc>

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

5.2. <get-time> Example

The following example is similar to the one presented in Section 5.1.,
except that in this example the client includes a <get-time>
element in its RPC, and the server consequently responds with an
<execution-time> element (as shown in Figure 2).

<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <get-time
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-time">
    </get-time>
  </config>
</rpc>

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

The following example presents a scenario in which the scheduled-time is not within the scheduling tolerance, i.e., it is too far in the past, and therefore an rpc-error is returned.

```xml
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
    <execution-time>
        2015-10-21T04:29:00.235Z
    </execution-time>
</rpc-reply>
```
6. Security Considerations

The security considerations of the NETCONF protocol in general are discussed in [RFC6241].

The usage of the time capability defined in this document can assist an attacker in gathering information about the system, such as the exact time of future configuration changes. Moreover, the time elements can potentially allow an attacker to learn information about the system's performance. Furthermore, an attacker that sends malicious RPC messages can use the time capability to amplify her attack; for example, by sending multiple RPC messages with the same scheduled time. It is important to note that the security measures described in [RFC6241] can prevent these vulnerabilities.

The time capability relies on an underlying time synchronization protocol. Thus, an attack against the time protocol can potentially compromise NETCONF when using the time capability. A detailed discussion about the threats against time protocols and how to mitigate them is presented in [TimeSec].

7. IANA Considerations

This document proposes to register the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

    urn:ietf:params:netconf:capability:time:1.0

This document proposes to register the following XML namespace URN in the 'IETF XML registry', following the format defined in [RFC3688]:

This document proposes to register a module name in the 'YANG Module Names' registry, defined in [RFC6020].

name: ietf-netconf-time
prefix: nct

RFC: TBD

8. Acknowledgments

The authors gratefully acknowledge Joe Marcus Clarke, Andy Bierman, Balazs Lengyel, Jonathan Hansford, Alon Schneider and Eylon Egozi for their insightful comments.

This work was supported in part by Israel Science Foundation grant ISF 1520/11.

This document was prepared using 2-Word-v2.0.template.dot.

9. References

9.1. Normative References


9.2. Informative References


Appendix A. YANG Module for the Time Capability

This section is normative.

<CODE BEGINS> file "ietf-netconf-time@2014-06-26.yang"

module ietf-netconf-time {

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

    prefix nct;

Mizrahi, Moses Expires December 26, 2014 [Page 17]
import ietf-netconf { prefix nc; }
import ietf-yang-types { prefix yang; }
import ietf-netconf-monitoring { prefix ncm; }

contact
  "Editor: Tal Mizrahi
  <dew@tx.technion.ac.il>
  Editor: Yoram Moses
  <moses@ee.technion.ac.il>";

description
  "This module defines a capability-based extension to the
  Network Configuration Protocol (NETCONF) that allows
  time-triggered configuration and management operations.
  This extension allows NETCONF clients to invoke configuration
  updates according to scheduled times, and allows NETCONF
  servers to attach timestamps to the data they send to NETCONF
  clients.";

revision 2014-06-26 {
  description
    "Initial version.";
  reference
    "draft-mm-netconf-time-capability:
      Time Capability in NETCONF";
}

typedef time-interval {
  type string {
    pattern '\d{2}:\d{2}:\d{2}(\.\d+)?';
  }
}

grouping scheduling-tolerance-parameters {
  description
    "Contains the parameters of the scheduling tolerance.";

  leaf sched-max-future {
    description
      "When the scheduled time is in the future, i.e., greater
than the present time, this leaf defines the maximal
difference between the scheduled time
and the present time that the server is willing to
accept. If the difference exceeds this number, the
server responds with an error.";
type time-interval;
default 00:00:15.0;
}

leaf sched-max-past {
  description
  "When the scheduled time is in the past, i.e., less
  than the present time, this leaf defines the maximal
difference between the present time
  and the scheduled time that the server is willing to
  accept. If the difference exceeds this number, the
  server responds with an error.";
type time-interval;
default 00:00:15.0;
}

// extending the get-config operation
augment /nc:get-config/nc:input {
  description
  "Adds the time element to <get-config>.";

  leaf scheduled-time {
    description
    "The time at which the RPC is scheduled to be performed.";
type yang:date-and-time;
  }

  leaf get-time {
    description
    "Indicates that the rpc-reply should include the
    execution-time.";
type empty;
  }
}
augment /nc:get-config/nc:output {
  description
      "Adds the time element to <get-config>.";
  leaf execution-time {
      description
          "The time at which the RPC was executed.";
      type yang:date-and-time;
  }
}

augment /nc:get/nc:input {
  description
      "Adds the time element to <get>.";
  leaf scheduled-time {
      description
          "The time at which the RPC is scheduled to be performed.";
      type yang:date-and-time;
  }
  leaf get-time {
      description
          "Indicates that the rpc-reply should include the
execution-time.";
      type empty;
  }
}

augment /nc:get/nc:output {
  description
      "Adds the time element to <get>.";
  leaf execution-time {
      description
          "The time at which the RPC was executed.";
      type yang:date-and-time;
  }
}
augment /nc:copy-config/nc:input {
    description
    "Adds the time element to <copy-config>.";

    leaf scheduled-time {
        description
        "The time at which the RPC is scheduled to be performed.";
        type yang:date-and-time;
    }

    leaf get-time {
        description
        "Indicates that the rpc-reply should include the execution-time.";
        type empty;
    }
}

augment /nc:copy-config/nc:output {
    description
    "Adds the time element to <copy-config>.";

    leaf execution-time {
        description
        "The time at which the RPC was executed.";
        type yang:date-and-time;
    }
}

augment /nc:edit-config/nc:input {
    description
    "Adds the time element to <edit-config>.";

    leaf scheduled-time {
        description
        "The time at which the RPC is scheduled to be performed.";
        type yang:date-and-time;
    }

    leaf get-time {
description
  "Indicates that the rpc-reply should include the
  execution-time.";
  type empty;
}
}

augment /nc:edit-config/nc:output {
  description
  "Adds the time element to <edit-config>.";

  leaf execution-time {
    description
    "The time at which the RPC was executed.";
    type yang:date-and-time;
  }
}

augment /nc:delete-config/nc:input {
  description
  "Adds the time element to <delete-config>.";

  leaf scheduled-time {
    description
    "The time at which the RPC is scheduled to be performed.";
    type yang:date-and-time;
  }

  leaf get-time {
    description
    "Indicates that the rpc-reply should include the
    execution-time.";
    type empty;
  }
}

augment /nc:delete-config/nc:output {
  description
  "Adds the time element to <delete-config>.";

  leaf execution-time {
augment /nc:lock/nc:input {
    description "Adds the time element to <lock>.";
    leaf scheduled-time {
        description "The time at which the RPC is scheduled to be performed.";
        type yang:date-and-time;
    }
    leaf get-time {
        description "Indicates that the rpc-reply should include the execution-time.";
        type empty;
    }
}

augment /nc:lock/nc:output {
    description "Adds the time element to <lock>.";
    leaf execution-time {
        description "The time at which the RPC was executed.";
        type yang:date-and-time;
    }
}

augment /nc:unlock/nc:input {
    description "Adds the time element to <unlock>.";
    leaf scheduled-time {
        description "The time at which the RPC was executed.";
        type yang:date-and-time;
    }
}
"The time at which the RPC is scheduled to be performed.";
   type yang:date-and-time;
}
leaf get-time {
   description
   "Indicates that the rpc-reply should include the
   execution-time.";
   type empty;
}

augment /nc:unlock/nc:output {
   description
   "Adds the time element to <unlock>.";
   leaf execution-time {
   description
   "The time at which the RPC was executed.";
   type yang:date-and-time;
   }
}

augment /nc:close-session/nc:input {
   description
   "Adds the time element to <close-session>.";
   leaf scheduled-time {
   description
   "The time at which the RPC is scheduled to be performed.";
   type yang:date-and-time;
   }
   leaf get-time {
   description
   "Indicates that the rpc-reply should include the
   execution-time.";
   type empty;
   }
}
augment /nc:close-session/nc:output {
  description
  "Adds the time element to <close-session>.";

  leaf execution-time {
    description
    "The time at which the RPC was executed.";
    type yang:date-and-time;
  }
}

augment /nc:kill-session/nc:input {
  description
  "Adds the time element to <kill-session>.";

  leaf scheduled-time {
    description
    "The time at which the RPC is scheduled to be performed.";
    type yang:date-and-time;
  }

  leaf get-time {
    description
    "Indicates that the rpc-reply should include the execution-time.";
    type empty;
  }
}

augment /nc:kill-session/nc:output {
  description
  "Adds the time element to <kill-session>.";

  leaf execution-time {
    description
    "The time at which the RPC was executed.";
    type yang:date-and-time;
  }
}

augment /nc:commit/nc:input {
add the time element to <commit>.

leaf scheduled-time {  
    description  
        "The time at which the RPC is scheduled to be performed.";
    type yang:date-and-time;
}

leaf get-time {  
    description  
        "Indicates that the rpc-reply should include the  
            execution-time.";
    type empty;
}

augment /nc:commit/nc:output {  
    description  
        "Adds the time element to <commit>.";
    leaf execution-time {  
        description  
            "The time at which the RPC was executed.";
        type yang:date-and-time;
    }
}

augment /ncm:netconf-state {  
    container scheduling-tolerance {  
        description  
            "The scheduling tolerance when the time capability  
                is enabled.";
        uses scheduling-tolerance-parameters;
    }
}

rpc cancel-schedule {  
    description  
        "Cancels a scheduled message.";
    reference  

input {
  leaf cancelled-message-id {
    description "The ID of the message to be cancelled.";
    type string;
  }
  leaf get-time {
    description "Indicates that the rpc-reply should include
    the execution-time.";
    type empty;
  }
}
}
output {
  leaf execution-time {
    description "The time at which the RPC was executed.";
    type yang:date-and-time;
  }
}
}

notification netconf-scheduled-message {
  description "Indicates that a scheduled message was received.";
  reference "draft-mm-netconf-time-capability:
  Time Capability in NETCONF";
  leaf scheduled-message-id {
    description "The ID of the scheduled message.";
    type string;
  }
  leaf scheduled-time {
    description "The time at which the RPC is scheduled to be performed.";
  }
}
type yang:date-and-time;
}

}<CODE ENDS>

Authors’ Addresses

Tal Mizrahi
7/43 Gotl Levin st.
Haifa, 3292207, Israel

Email: dew@tx.technion.ac.il

Yoram Moses
Department of Electrical Engineering
Technion - Israel Institute of Technology
Technion City, Haifa, 32000, Israel

Email: moses@ee.technion.ac.il