

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 23, 2014

A. Bierman
YumaWorks, Inc.
April 21, 2014

NETCONF Efficiency Extensions
draft-bierman-netconf-efficiency-extensions-01

Abstract

This document describes protocol extensions to improve the efficiency of the Network Configuration Protocol (NETCONF). Protocol capabilities and operations are defined to reduce network usage and transaction complexity.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on October 23, 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 (<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.

Internet-Draft

NETCONF-EX

April 2014

Table of Contents

1.	Introduction	4
1.1.	Terminology	4
1.1.1.	NETCONF	4
1.1.2.	YANG	4
1.1.3.	RESTCONF	5
1.1.4.	YANG Patch	5
1.1.5.	Terms	5
1.1.6.	Tree Diagrams	5
1.2.	Problem Statement	6
1.2.1.	Initial Configuration Retrieval	6
1.2.2.	Message Encoding	6
1.2.3.	Datastore Editing	7
1.2.4.	Data Retrieval	8
1.3.	Solution	10
1.3.1.	Configuration ID Advertisement	10
1.3.2.	Message Encoding Negotiation	10
1.3.3.	<edit2> Operation	11
1.3.4.	<get2> Operation	11
2.	Definitions	12
2.1.	"config-id" Capability	12
2.1.1.	Overview	12
2.1.2.	Dependencies	13
2.1.3.	Capability Identifier	13
2.1.4.	New Operations	14
2.1.5.	Modifications to Existing Operations	14
2.1.6.	Interactions with Other Capabilities	14
2.2.	"encoding" Capability	14
2.2.1.	Overview	14
2.2.2.	Dependencies	16
2.2.3.	Capability Identifier	16
2.2.4.	New Operations	17
2.2.5.	Modifications to Existing Operations	17
2.2.6.	Interactions with Other Capabilities	17
2.3.	<edit2> Protocol Operation	17
2.3.1.	<edit2> Input	17
2.3.2.	<edit2> Output	18
2.3.3.	<edit2> YANG Tree Diagram	19
2.3.4.	<edit2> Example	20
2.4.	<complete-commit> Operation	22
2.4.1.	<complete-commit> Input	22
2.4.2.	<complete-commit> Output	22

2.4.3.	<complete-commit> YANG Tree Diagram	23
2.4.4.	<complete-commit> Example	23
2.5.	<revert-commit> Operation	23
2.5.1.	<revert-commit> Input	23
2.5.2.	<revert-commit> Output	23

2.5.3.	<revert-commit> YANG Tree Diagram	24
2.5.4.	<revert-commit> Example	24
2.6.	<get2> Protocol Operation	24
2.6.1.	Depth Filters	24
2.6.2.	Time Filters	25
2.6.3.	<get2> Input	25
2.6.4.	<get2> Output	26
2.6.5.	<get2> YANG Tree Diagram	27
2.6.6.	<get2> Example	27
2.7.	NETCONF-EX YANG Module	28
2.8.	XSD for NETCONF-EX Metadata	44
3.	IANA Considerations	47
3.1.	NETCONF-EX XML Namespace	47
3.2.	NETCONF-EX XML Schema	47
3.3.	NETCONF-EX YANG Module	47
4.	Security Considerations	48
5.	Change Log	49
5.1.	00 to 01	49
5.1.1.	Removed :capability-id Capability	49
5.1.2.	RESTCONF Alignment	49
6.	Normative References	50
Appendix A.	Open Issues	52
A.1.	resource-identifier-type	52
A.2.	no YANG for top-level message nodes	52
A.3.	config-id attribute	52
A.4.	<get2> nodeset retrieval	52
Appendix B.	Additional Examples	53
B.1.	YANG Module Used in Examples	53
B.2.	YANG Data Used in Examples	54
B.3.	<edit2> Examples	55
B.3.1.	Confirmed Commit on the "running" Datastore	55
B.3.2.	Conditional Editing with "if-match" Parameter	56
B.3.3.	Bulk Editing with "target-resource" Parameter	59
B.3.4.	Edit Validation with "test-only" Parameter	61
B.4.	<get2> Examples	62
B.4.1.	If-Modified-Since Non-Empty Filter Retrieval	62

B.4.2.	If-Modified-Since Empty Filter Retrieval	64
B.4.3.	Keys Only Filter Retrieval	64
B.4.4.	Test for Node Existence with Depth=1	66
B.4.5.	Retrieve Only Non-Configuration Data Nodes	67
Author's Address		69

[1.](#) Introduction

There is a need for standard mechanisms to allow NETCONF [[RFC6241](#)] application designers to manage NETCONF servers more efficiently when used in network environments with poor connectivity, low bandwidth, and/or high latency. In such conditions, it is desirable to minimize network usage wrt/ the size of protocol messages and the number of protocol operations required to perform a network management function.

[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](#)]:

- o candidate configuration datastore
- o client
- o configuration data
- o datastore

- o configuration datastore
- o protocol operation
- o running configuration datastore
- o server
- o startup configuration datastore

[1.1.2.](#) YANG

The following terms are defined in [[RFC6020](#)]:

- o container
- o data node

- o key leaf
- o leaf
- o leaf-list
- o list

[1.1.3.](#) RESTCONF

The following terms are defined in [[RESTCONF](#)]:

- o data resource
- o datastore resource

[1.1.4.](#) YANG Patch

The following term is defined in [[YANG-Patch](#)]:

- o YANG Patch

[1.1.5.](#) Terms

The following terms are defined:

- o config ID: An opaque string identifier that represents the state of the running datastore contents on the server. A new config ID is chosen by the server each time the server running configuration datastore is altered in any way.
- o depth filter: A mechanism implemented within the NETCONF server to allow a client to retrieve only a limited number of levels within the a subtree, instead of retrieving the entire subtree.
- o time filter: A mechanism implemented within the NETCONF server to allow a client to retrieve only data that has been modified since a specified data and time.

[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:

- o Brackets "[" and "]" enclose list keys.

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

[1.2.](#) Problem Statement

This document attempts to address the following problems with NETCONF

protocol procedures.

[1.2.1.](#) Initial Configuration Retrieval

A client application often needs to retrieve the entire running configuration datastore contents, usually at the start of an editing session. The `<rpc-reply>` for this `<get-config>` request can be very large (e.g., greater than 250,000 bytes).

If a large number of server connections are lost and then restarted, the quantity of large `<rpc-reply>` messages from every server could severely impact network performance.

It would be useful if the `<hello>` message exchange could be enhanced so an entity-tag value for the current running datastore configuration is included in the server `<hello>` message. A client can cache the server configuration identifier and omit an initial `<get-config>` operation if the value from the server `<hello>` message matches the cached value.

[1.2.2.](#) Message Encoding

NETCONF uses a hard-wired message encoding format, namely XML. However, XML tends to be verbose, especially for YANG data models that have long data node identifiers.

There is no reason for the NETCONF message encoding to be hardwired, except for the `<hello>` message. It would be useful if the NETCONF protocol could support other message encoding formats, such as JSON [[JSON](#)]. The `<hello>` message exchange could be enhanced so the client and server negotiated the message encoding to use for all other messages via an capability exchange included in both `<hello>` messages.

[1.2.3.](#) Datastore Editing

There are several deficiencies with the NETCONF editing procedures that could be improved.

Multi-operation functions can be required. A single edit can take up to 9 operations. Several operations are required to complete a set of 1 or more edits on a NETCONF server. Each operation uses 1

request and 1 response message. If the candidate datastore is used, then 1 extra operation is required (for the <commit> operation) to activate the edit(s). If the startup datastore is used then 1 extra operation is required (for the <copy-config> operation) to save the running datastore contents in non-volatile storage. If global locking is used, then 2 extra operations are required for each datastore involved (candidate, running, startup) Since the datastore is locked at the start and unlocked at the end of the entire edit operation, these extra roundtrip times are intervals in which the datastore is being locked, but no datastore access is being done.

Obtaining locks can be expensive. If the server has more than 1 datastore (e.g., candidate + running or running + startup), then multiple lock requests are required, since the <lock> and <unlock> operations on affect 1 datastore at a time. This can cause a long delay or even deadlock if multiple clients are attempting to obtain global locks at once. E.g., client 1 holds a lock on the candidate datastore and is trying to lock the running datastore. At the same time, client 2 holds a lock on the running datastore and is trying to lock the candidate datastore.

Using locks can be brittle. NETCONF clients are intended to be programmatic, so is not likely that locks will be long-lived. Global locks are designed to be short-lived since they block write access to the entire datastore. If lock collisions do occur, they are likely to be cleared very quickly. It would be useful if the client could request how long to wait for locks to clear instead of immediately rejecting an edit request due to an 'in-use' error.

Edit operations are implied by <config> content. NETCONF uses a default operation and explicit operation attribute within an arbitrarily complete XML subtree to represent a configuration datastore. There are several corner-cases that are not standardized, and very implementation-dependent:

- order the edits are processed
- handling of nested operation attributes
- handling of duplicate subtrees
- error handling (code points, number of errors, etc.)
- move operations are not explicit and can be interpreted as a request to remove and re-add an entry, not just move user-ordered data

Edit operations are not protected against multi-client alterations. It is a simple and common practice to retrieve a configuration data resource, changing 1 or more fields, and then update the resource on the server. Since retrieval and edit operations are separate there is always a chance that another client has altered the resource after the <get-config> operation, but before the <edit-config> operation, by the first client. Each client could be protected if there was an entity tag associated with each data resource, and an edit request could be rejected if the client attempted to edit a different version of the data resource than expected.

There is no bulk-edit support. If the same edit is needed in multiple instances of a particular data resource, then the data must be repeated for each instance in the <edit-config> or <copy-config> request. The request message size could be minimized if there was a way to apply a set of edits to multiple target nodes at once.

There is no confirmed commit support for the running datastore. The ability to backup the running datastore, change it, and revert it unless the client confirms the changes has nothing to do with the candidate datastore. A NETCONF server with limited memory is not likely to support the candidate datastore. This feature is useful for any type of network-wide configuration change, regardless of device size.

[1.2.4.](#) Data Retrieval

NETCONF data retrieval via the <get> and <get-config> operations can be very inefficient. Some vendors do not even support <get> because it can be such a resource-intensive operation and return an enormous amount of data, especially if all server data is requested at once.

A client cannot retrieve just the non-configuration data. The NETCONF <get> operation allows a client to retrieve data from the server but it returns all data, including configuration datastore nodes. The <get-config> operation already returns all configuration datastore nodes.

It was originally thought that <get> should return all nodes so the

client would not have to correlate configuration and non-configuration data nodes, since they would be mixed together in the reply. Operational experience has shown that the <get> operation without reasonable filters to reduce the returned data can significantly degrade device performance and return enormous XML instance documents in the <rpc-reply>.

There is no "last-modified" indication or time filtering. The NETCONF protocol has no standard mechanisms to indicate to a client when a datastore was last modified, or to allow a client to retrieve data only if it has been modified since a specified time. This makes polling applications very inefficient because they will regularly burden the server and the network and themselves with retrieval and processing requests for data that has not changed.

There is no simple list instance discovery mechanism. Sometimes the client application wants to discover what data exists on the server, particularly list entries. There is a need for a simple mechanism to retrieve just the key leaf nodes within a subtree. The NETCONF subtree filtering mechanism does provide a very complex way for the client to request just key leafs for specific list entries. A simpler mechanism is needed which will allow the client to discover the list instances present.

There is no subtree depth control. NETCONF filters allow the client to select specific sub-trees within the conceptual datastore on the server. However, sometimes the client does not really need the entire subtree, which may contain many nested list entries, and be very large. There is sometimes a need to limit the depth of the sub-trees retrieved from the server. A consistent and simple algorithm for determining what data nodes start a new level is needed.

The content filter specification is not extensible. The NETCONF <get> and <get-config> operations use a hard-coded content filtering mechanism. They use a "type" XML attribute to indicate which of two filter specification types they support, and a "select" XML attribute if the :xpath capability is supported and an XPath [[XPATH](#)] expression filter specification is provided.

This design does not allow additional content filter specification types to be supported by an implementation. It does not allow the standard to be easily extended in a modular fashion. In addition, this design does not allow YANG statements to be used to properly describe the protocol operation. The special "get-filter-element-attributes" YANG extension in the ietf-netconf module is not extensible, and it does not really count as proper

YANG, since this extension is outside the YANG language definition.

There is no standard metadata or standard way to retrieve metadata. The <with-defaults> parameter allows 1 specific type of metadata to be returned (i.e., 'report-all-tagged' mode). This ad-hoc approach does not scale well and is not extensible. It would be useful if standard and vendor-specific metadata could be identified and retrieved with standard operations.

[1.3.](#) Solution

This document defines some NETCONF protocol operations and new capabilities to reduce network usage and increase functionality at the same time.

All NETCONF efficiency extensions are completely backward-compatible with the current definitions in [\[RFC6241\]](#). An old client will ignore any new <capability> URIs sent by the server, and will not use the new operations. No existing operations are affected by the new operations, so the extensions will be transparent to an existing NETCONF client.

[1.3.1.](#) Configuration ID Advertisement

A new capability called "config-id" is defined to identify the current running datastore configuration contents with an opaque string. A client can cache this value for each server that supports this capability, along with a copy of its running configuration. When a new session is started, the client can examine the "config-id" <capability> URI sent by the server. If it is the same as the cached value then the client can use the cached running datastore copy instead of sending an initial <get-config> operation to the server. The :config-id capability is ignored in the calculation of the :capability-id capability. Refer to [Section 2.1](#) for details on configuration ID advertisement.

[1.3.2.](#) Message Encoding Negotiation

A new capability called "encoding" is defined to allow a client to request that an alternate message encoding be used for the NETCONF session. The capability is encoded as a comma-separated list of

media types. This list is ordered by the client in the order of highest preference first. The server list is unordered. The first match (done in client priority) is the message encoding used for the rest of session. Refer to [Section 2.2](#) for details on message encoding negotiation.

[1.3.3.](#) <edit2> Operation

A new NETCONF protocol operation called <edit2> is defined to address the deficiencies described in [Section 1.2.3](#). This operation allows the entire NETCONF edit procedure to be accomplished with 1 request message. The editing procedures are aligned with the resource model defined in [\[RESTCONF\]](#). Refer to [Section 2.3](#) for details on <edit2> operation.

The "confirmed-commit" procedure has been integrated into the <edit2> operation, and can be supported by any server without requiring support for the candidate datastore. It is optional to implement, based on the "confirmed-edit" capability defined in [Section 2.7](#).

Refer to [Section 2.4](#) for details on the <complete-commit> operation and [Section 2.5](#) for details on the <revert-commit> operation.

[1.3.4.](#) <get2> Operation

A new NETCONF protocol operation called <get2> is defined to address the deficiencies described in [Section 1.2.4](#). This operation allows several filter types to be combined to control the data that is returned in the <rpc-reply> message, and an extensible framework for retrieving metadata associated with datastore or data resources. Refer to [Section 2.6](#) for details on <get2> operation.

[2.](#) Definitions

This section defines the NETCONF efficiency extensions:

- :config-id Capability
- :encoding Capability
- <edit2> Operation
- <complete-commit> Operation
- <revert-commit> Operation
- <get2> Operation

[2.1.](#) "config-id" Capability

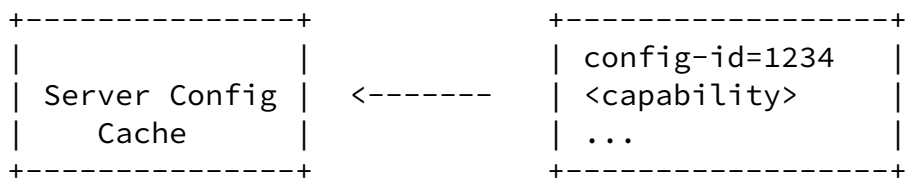
[2.1.1.](#) Overview

The :config-id capability indicates that the server maintains a config ID for the running configuration datastore. This identifier value is selected by the server and treated as an opaque string by the client.

- 1) Client keeps a cache of server configurations.
- 2) Server always sends its current config-id value in the "config-id" <capability> URI.

Client

Server X <hello> message



3) Client checks cache for server X, config-id=1234.
 If found, then OK to use the cached configuration copy.
 If not found, then send a <get-config> for the running configuration to create or update the cached copy.

The server SHOULD save the config ID for the running datastore in non-volatile storage. When the server boots or restarts, the initial configuration ID SHOULD be the same as the last instantiation, if the server does not support the :startup capability (so the non-volatile stored version mirrors the running datastore). If the server does support the :startup capability, then the initial configuration ID SHOULD be the same as the version last saved to non-volatile storage.

[2.1.1.1.](#) :config-id Capability Example

The :config-id capability is sent in every server <hello> message. The "id" parameter for the :config-id capability is set to the current config ID for the running datastore on the server:

```

# Server starts session 3 with an abbreviated <hello>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:netconf:capability:config-id?id=4284
    </capability>
    // ... rest of <capability> elements
  </capabilities>
  <session-id>3</session-id>
</hello>

```

[2.1.2.](#) Dependencies

The :config-id capability is not dependent on any other capabilities.

[2.1.3.](#) Capability Identifier

The :config-id capability is identified by the following capability string:

urn:ietf:params:netconf:capability:config-id:1.0

This capability MUST be advertised in every server <hello> message. The :config-id capability URI MUST contain an "id" argument assigned an opaque string value indicating the current config ID value for the running datastore. For example:

urn:ietf:params:netconf:capability:config-id:1.0?id=6882391

The current config ID value MUST be updated any time a "netconf-config-change" event would be generated by the server.

If [[RFC6470](#)] is supported, then the "config-id" leaf defined in [Section 2.7](#) MUST be included in <netconf-config-change> event notifications.

If the "with-metadata" parameter in the <get2> operation specifies the "config-id" identity, then the server MUST return the current config ID for the running datastore, if the "source" parameter identifies the running datastore. The server MAY maintain config IDs for other datastores as well.

[2.1.4.](#) New Operations

The :config-id capability does not introduce any new protocol operations.

[2.1.5.](#) Modifications to Existing Operations

The :config-id capability does not modify any existing protocol operations.

[2.1.6.](#) Interactions with Other Capabilities

The :config-id capability does not interact with any other capabilities.

[2.2.](#) "encoding" Capability

[2.2.1.](#) Overview

The :encoding capability is used by the client to request an alternate message encoding be used instead of XML. The client and server both send a list of media types for the message encodings they support, encoded as a comma-separated list (with no whitespace). The client list is an ordered by preference. The server list is unordered.

+-----+		+-----+	+-----+
<hello>	Client and	<rpc>	<rpc-reply>
	server select		
+-----+	protocol base	+-----+	+-----+
	and encoding		

Always encoded
in XML w/base:1.0
message framing

--- > Start encoding all messages
in the selected encoding
and message framing

Both the client and server will examine the others <hello> message for the "encoding" <capability> URI. If not present, then the default encoding is used, which is XML.

The client list is compared against the server list, checked in the client specified order. If the same media type appears in the server list, then that is the encoding that will be active for the remainder of the session (i.e., starting with the first <rpc> request). All <rpc>, <rpc-reply>, and <notification> messages MUST be encoded in the negotiated encoding.

Both the client and server MUST support the "application/xml" media type to be backward-compatible with [\[RFC6241\]](#).

If "application/json" encoding is used, then the encoding defined in

[[I-D.lhotka-netmod-json](#)] MUST be used so namespaces will be properly identified. Any metadata that needs to be encoded MUST be encoded according to the procedure defined in [[RESTCONF](#)], section 4.4.

The message framing used for the session is unaffected by this capability. The "base1.0" vs. "base1.1" negotiation defined in [[RFC6241](#)] determines the message framing that is used for the entire session.

[2.2.1.1](#). :encoding Capability Example

In this example, the client supports the following message encodings, shown in the preferred order.

- Efficient XML Interchange (EXI)
- JSON
- XML

Some extra whitespace has been added for display purposes only.

```
# Client requests a new session with alternate encoding
# also requesting an abbreviated hello
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>
      urn:ietf:params:netconf:capability:encoding:1.0?
      types=application/exi,application/json,application/xml
    </capability>
  </hello>
```

The server supports the following encodings:

- XML
- JSON

Since the most preferred media type in common is "application/json", the JSON encoding is used for the remainder of the session.

In this example, the server sends an full <hello> message to the client, truncated for brevity. Extra whitespace has been added for display purposes only.


```
# Server starts session 4 with JSON encoding
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>
      urn:ietf:params:netconf:capability:encoding:1.0?
      types=application/xml,application/json
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:config-id?id=2130
    </capability>
    <!-- rest of URIs ... -->
  </capabilities>
  <session-id>4</session-id>
</hello>
```

At this point, both the client and server switch to JSON encoding:

```
# client send a <kill-session> request
{ "ietf-netconf:rpc" : {
  "@message-id" : "201",
  "kill-session" : {
    "session-id" : 42
  }
}
}

# server sends an <ok/> reply
{ "ietf-netconf:rpc-reply" : {
  "@message-id" : "201",
  "ok" : [null]
}
}
```

[2.2.2.](#) Dependencies

The :encoding capability is not dependent on any other capabilities.

[2.2.3.](#) Capability Identifier

The :encoding capability is identified by the following capability string:

```
urn:ietf:params:netconf:capability:encoding:1.0
```

This capability MUST be advertised in every server <hello> message.

The "encoding" capability URI MUST contain a "types" argument

containing a comma-separated list of media types that represent the message encoding formats supported by the server.

If the client supports the :encoding capability, it SHOULD include an "encoding" <capability> URI in its <hello> message. The client MAY omit this capability if XML encoding is desired.

For example (line wrapped for display purposes only)

```
urn:ietf:params:netconf:capability:encoding:1.0?  
types=application/json,application/xml
```

[2.2.4.](#) New Operations

The :encoding capability does not introduce any new protocol operations.

[2.2.5.](#) Modifications to Existing Operations

The :encoding capability does not modify any existing protocol operations.

[2.2.6.](#) Interactions with Other Capabilities

The :encoding capability does not interact with any other capabilities.

[2.3.](#) <edit2> Protocol Operation

The <edit2> operation is specified with a YANG "rpc" statement, defined in [Section 2.7](#). This operation allows the entire NETCONF transaction procedure to be performed in a single operation or multiple operations, depending on the input parameters used.

There are no XML attributes used (e.g., "operation" from [RFC 6241](#), "insert", "value" from [RFC 6020](#)). Instead, configuration edits are specified with an edit list, using the YANG Patch mechanism defined in [\[RESTCONF\]](#). This is used instead of a complete XML instance document, e.g. <config> element, to represent an unordered patch list inferred from the diffs. (Although YANG Patch can be used in this

mode if client wants to merge or replace the entire configuration datastore).

[2.3.1.](#) <edit2> Input

- o target: name of the configuration datastore being edited

- o target-resource: XPath node-set expression representing 1 or more target resources within the datastore to edit.
- o yang-patch: container of ordered edits to apply to the target resource(s).
- o test-only: flag to request that the edit request be validated but no edits should actually be applied
- o if-match: if the entity tag for the target resource(s) does not exactly match the supplied value then the edit request is rejected.
- o with-locking: if present then the server will provide exclusive write access to this <edit2> operation and possible confirmed-commit procedure.
- o max-lock-wait: amount of time the client is willing to wait for locks to clear, if "with-locking" parameter is present.
- o activate-now: if present and the target is the candidate datastore, then an implicit <commit> operation will be performed if the edit operation is successfully applied.
- o nvstore-now: if present and the server supports the startup datastore, and the edits have been activated in the running datastore, then an implicit <copy-config> operation (from the running to the startup datastore) will be attempted by the server.
- o confirmed: request that a confirmed commit be started or extended.
- o confirm-timeout: the amount of time for the server to wait for an <edit2> request that extends, a <complete-commit> request to

finish, or a <revert-commit> request to cancel a confirmed commit procedure in progress.

- o persist: identifier string to use in the "persist-id" parameter to extend, complete, or cancel a confirmed commit procedure.
- o persist-id: identifier string to extend a confirmed commit procedure in progress.

[2.3.2.](#) <edit2> Output

Positive Response:

This operation returns data containing a "yang-patch-status" report (defined in [[RESTCONF](#)]) instead of an "ok" element. This report

Bierman

Expires October 23, 2014

[Page 18]

Internet-Draft

NETCONF-EX

April 2014

contains an "ok" element that is present if the entire operation succeeded.

Error Response:

The <rpc-error> element can be returned, e.g., if the message contains invalid parameter syntax. The server MUST report editing errors in the "edit" list within the "yang-patch-status" container.

[2.3.3.](#) <edit2> YANG Tree Diagram

Key: DRI = data-resource-identifier

```
+---x edit2
  +--ro input
    | +--ro target
    | | +--ro (datastore-target)
    | | | +--:(candidate)
    | | | | +--ro candidate?    empty
    | | | +--:(running)
    | | | | +--ro running?      empty
    | +--ro target-resource?    yang:xpath1.0
    +--ro yang-patch
      | +--ro patch-id?    string
      | +--ro comment?     string
```



```

| | +--ro edit [edit-id]
| |   +--ro edit-id      string
| |   +--ro operation    enumeration
| |   +--ro target       data-resource-identifier
| |   +--ro point?       data-resource-identifier
| |   +--ro where?       enumeration
| |   +--ro value
| +--ro test-only?       empty
| +--ro if-match?        yang-entity-tag
| +--ro with-locking?    empty
| +--ro max-lock-wait?   uint32
| +--ro activate-now?    empty
| +--ro nvstore-now?     empty
| +--ro confirmed?       empty
| +--ro confirm-timeout? uint32
| +--ro persist?         string
| +--ro persist-id?      string
+--ro output
  +--ro yang-patch-status
    +--ro patch-id?      string
    +--ro (global-status)?
      | +--:(global-errors)

```

```

| | +--ro errors
| |   +--ro error
| |   +--ro error-type    enumeration
| |   +--ro error-tag     string
| |   +--ro error-app-tag? string
| |   +--ro error-path?   DRI
| |   +--ro error-message? string
| |   +--ro error-info
| |
| +--:(ok)
|   +--ro ok?             empty
+--ro edit-status
  +--ro edit [edit-id]
    +--ro edit-id string
    +--ro (edit-status-choice)?
      +--:(ok)
      | +--ro ok?         empty
      +--:(errors)
        +--ro errors

```



```

+--ro error
  +--ro error-type      enumeration
  +--ro error-tag       string
  +--ro error-app-tag?  string
  +--ro error-path?     DRI
  +--ro error-message?  string
  +--ro error-info

```

[2.3.4.](#) <edit2> Example

In this example, an "all-in-one" YANG Patch edit is shown. the following conditions apply:

- The server supports the :candidate and :startup capabilities
- The "example-ex" YANG module is supported by the server

The starting state of the "/forests" data structure is described in [Appendix B.2](#). The client is adding an "oak" tree and changing the location of the "birch" tree in the "north" forest.

```

<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource>
      /ex:forests/ex:forest[ex:name='north']
    </target-resource>
    <yang-patch>
      <patch-id>north-forest-patch</patch-id>
      <comment>
        Add an oak tree and change location of the birch tree

```



```

</comment>
<edit>
  <edit-id>oak</edit-id>
  <operation>create</operation>
  <target>/ex:trees</target>
  <value>
    <ex:tree>
      <ex:name>oak</ex:name>
      <ex:location>hillside</ex:location>
    </ex:tree>
  </value>
</edit>
<edit>
  <edit-id>birch</edit-id>
  <operation>merge</operation>
  <target>/ex:trees/ex:tree/birch</target>
  <value>
    <ex:location>west valley</ex:location>
  </value>
</edit>
</yang-patch>
<activate-now/>
<nvstore-now/>
</edit2>
</rpc>

```

The edit succeeds, and the "yang-patch-status" container is returned to the client with the <ok/> status for both tree edits:

```

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">

```



```
<patch-id>north-forest patch</patch-id>
<ok/>
<edit-status>
  <edit>
    <edit-id>oak</edit-id>
    <ok/>
  </edit>
  <edit>
    <edit-id>birch</edit-id>
    <ok/>
  </edit>
</edit-status>
</yang-patch-status>
</rpc-reply>
```

Refer to [Appendix B.3](#) for additional <edit2> protocol operation examples.

[2.4.](#) <complete-commit> Operation

A new NETCONF protocol operation called <complete-commit> is defined to complete a confirmed commit procedure.

[2.4.1.](#) <complete-commit> Input

There is one optional parameter for this protocol operation:

- o persist-id: an identifier string that MUST match the "persist" value, if it was used in the confirmed-commit procedure.

[2.4.2.](#) <complete-commit> Output

Positive Response:

When there is a confirmed-commit procedure in progress and it is successfully completed, then an <ok/> element is returned.

Negative Response: An <rpc-error> response is sent if the request cannot be completed for any reason.

[2.4.3.](#) <complete-commit> YANG Tree Diagram

```
+---x complete-commit
  +--ro input
    +--ro persist-id?   string
```

[2.4.4.](#) <complete-commit> Example

In this example, the client has previously started a confirmed commit procedure using the "persist" parameter set to the value "abcdef".

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <complete-commit
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <persist-id>abcdef</persist-id>
  </complete-commit>
</rpc>
```

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

[2.5.](#) <revert-commit> Operation

A new NETCONF protocol operation called <revert-commit> is defined to cancel a confirmed commit procedure and revert the running datastore. The <cancel-commit> operation in [[RFC6241](#)] cannot be used because it requires the implementation of the candidate capability.

[2.5.1.](#) <revert-commit> Input

There is one optional parameter for this protocol operation:

- o persist-id: an identifier string that MUST match the "persist" value, if it was used in the confirmed-commit procedure.

[2.5.2.](#) <revert-commit> Output

Positive Response:

If there is a confirmed-commit procedure in progress and it is successfully cancelled, and the running datastore successfully reverted, then an <ok/> element is returned.

Negative Response: An <rpc-error> response is sent if the request

Internet-Draft

NETCONF-EX

April 2014

cannot be completed for any reason.

[2.5.3.](#) <revert-commit> YANG Tree Diagram

```
+---x revert-commit
  +--ro input
    +---ro persist-id?   string
```

[2.5.4.](#) <revert-commit> Example

In this example, the client has previously started a confirmed commit procedure using the "persist" parameter set to the value "abcdef".

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <revert-commit
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <persist-id>abcdef</persist-id>
  </revert-commit>
</rpc>
```

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

[2.6.](#) <get2> Protocol Operation

The <get2> operation is specified with a YANG "rpc" statement, defined in [Section 2.7](#). A specific datastore is selected for the source of the retrieval operation. Several different types of filters are provided. Filters are combined in a conceptual "logical-AND" operation, and are optional to use by the client. Not all filtering mechanisms are mandatory-to-implement for the server.

[2.6.1.](#) Depth Filters

A depth filter indicates how many subtree levels should be returned in the <rpc-reply>. This filter is specified with the "depth" input parameter for the <get2> protocol operation. The default "0" indicates that all levels from the requested subtrees should be

returned.

A new level is started for each YANG data node within the requested subtree. All top level data nodes are considered to be child nodes (level 1) of a conceptual <config> root.

If no content filters are provided, then level 1 is considered to include all top-level data nodes within the source datastore. Otherwise only the levels in selected subtrees will be considered, and not any additional top-level data nodes.

If the depth requested is equal to "1", then only the requested data nodes (or top-level data nodes) will be returned. This mechanism can be used to detect the existence of containers and list entries within a particular subtree, without returning any of the descendant nodes.

Higher depth values indicates the number of descendant nodes to include in the response. For example, if the depth requested is equal to "2", then only the requested data nodes (or top-level data nodes) and their immediate child data nodes will be returned.

[2.6.2.](#) Time Filters

A time filter specifies that data should only be returned if the last-modified timestamp for the target datastore is more recent than the timestamp specified in the "if-modified-since" parameter.

If this feature is supported, then the server will maintain a "last-modified" timestamp for the running datastore. The server MAY support additional nested timestamps for data nodes within the datastore. The server MAY support timestamps for other datastores.

When a request containing the "if-modified-since" parameter is received, the server will compare that timestamp to the "last-modified" timestamp for the source datastore. If it is greater than the specified value then data may be returned (depending on other filters). If the datastore timestamp value is less than or equal to the specified value, then an empty <data> element will be returned in the <rpc-reply>.

If the "full-delta" parameter is present, and the server maintains

"last-modified" timestamps for any data nodes within the source datastore, then the same type of comparison will be done for the data node to determine if it should be included in the response. If no "last-modified" timestamp is maintained for a data node, then the server will use the "last-modified" timestamp for its nearest ancestor, or for the datastore itself if there are none.

[2.6.3.](#) <get2> Input

- o source: A container indicating the conceptual datastore for the retrieval request.

- o filter-spec: A choice indicating the content filter specification for the retrieval request.
- o keys-only: A leaf indicating that only the key leafs, combined with other filtering criteria, should be returned.
- o if-modified-since: A leaf indicating the time filter specification for the retrieval request, according to the procedures in [Section 2.6.2](#).
- o full-delta: If present and the "if-modified-since" parameter is also present, then the entire datastore will be filtered by last modification time, not just the entire datastore.
- o depth: A leaf indicating the subtree depth level for the retrieval request, according to the procedures in [Section 2.6.1](#).
- o with-defaults: A leaf indicating the type of defaults handling requested, according to procedures in [\[RFC6243\]](#).
- o with-metadata: A leaf-list indicating the specific metadata that the server should add to the response, such as "last-modified" or "etag", encoded in XML according to the schema in [Section 2.8](#).
- o with-locking: if present then the server will provide exclusive write access to this <get2> operation so the target datastore is not modified during the entire retrieval operation.

- o max-lock-wait: amount of time the client is willing to wait for locks to clear, if "with-locking" parameter is present.

[2.6.4.](#) <get2> Output

Positive Response: A <data> element is returned which contains the data corresponding to the input parameters specified in the request. The child nodes of the <data> container correspond to top-level YANG data nodes.

If the server supports the "timestamps" YANG feature, and the target is the running datastore, then a "last-modified" attribute SHOULD be included in the <rpc-reply> element.

Negative Response: An <rpc-error> response is sent if the request cannot be completed for any reason.

[2.6.5.](#) <get2> YANG Tree Diagram

```
+---x get2
  +--ro input
    |   +--ro source
    |   |   +--ro (datastore-source)?
    |   |   |   +--:(candidate)
    |   |   |   |   +--ro candidate?    empty
    |   |   |   +--:(running)
    |   |   |   |   +--ro running?      empty
    |   |   |   +--:(startup)
    |   |   |   |   +--ro startup?      empty
    |   |   |   +--:(url)
    |   |   |   |   +--ro url?          inet:uri
    |   |   |   +--:(operational)
    |   |   |   +--ro operational?     empty
    |   +--ro (filter-spec)?
    |   |   +--:(subtree-filter)
    |   |   |   +--ro subtree-filter
    |   |   +--:(xpath-filter)
    |   |   |   +--ro xpath-filter?    yang:xpath1.0
```



```

|   +---ro keys-only?           empty
|   +---ro if-modified-since?  yang:date-and-time
|   +---ro full-delta?         empty
|   +---ro depth?              uint32
|   +---ro with-defaults?      with-defaults-mode
|   +---ro with-metadata*      identityref
|   +---ro with-locking?       empty
|   +---ro max-lock-wait?      uint32
+---ro output
    +---ro data

```

[2.6.6.](#) <get2> Example

In this example, the retrieval the "forests" resource is shown. the following conditions apply:

- The server supports the :candidate and :startup capabilities
- The "example-ex" YANG module is supported by the server

The starting state of the "/forests" data structure is described in [Appendix B.2](#). The client is retrieving just the "forests" node, along with the "last-modified" and "etag" metadata for that node. The "config-id" for the datastore is also requested. Locking is requested (with a maximum lock wait time of 5 seconds), just to make sure the metadata does not change during the request.

```

<rpc message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
    <depth>1</depth>
    <with-metadata>ncex:timestamps</with-metadata>
    <with-metadata>ncex:etags</with-metadata>
    <with-metadata>ncex:config-id</with-metadata>
    <with-locking />
    <max-lock-wait>5</max-lock-wait>
  </get2>

```


</rpc>

The server has a "forests" node so this node is returned along with the requested metadata for the node. Note that the XML namespace for the "ncex" metadata is the XSD target namespace defined in [Section 2.8](#), not the YANG namespace URI defined in [Section 2.7](#).

```
<rpc-reply message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:m="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0"
    m:last-modified="2012-09-09T02:00:00Z"
    m:config-id="3aee5601">
    <forests xmlns="http://example.com/ns/example-ex"
      m:last-modified="2012-09-09T02:00:00Z"
      m:etag="3aee5601" />
    </data>
  </rpc-reply>
```

Refer to [Appendix B.4](#) for additional <get2> protocol operation examples.

[2.7](#). NETCONF-EX YANG Module

This module imports the "with-defaults-parameters" grouping from [\[RFC6243\]](#).

Several YANG features are imported from [\[RFC6241\]](#). These correspond to the NETCONF capabilities (e.g., candidate, url, startup, xpath) but defined as YANG features instead of URIs.

Some data types are imported from [\[RFC6991\]](#):

- date-and-time
- uri
- xpath1.0

Two YANG groupings are imported from [\[YANG-Patch\]](#):

- yang-patch

- yang-patch-status

One notification is augmented from [\[RFC6470\]](#).

- netconf-configuration-change

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

<CODE BEGINS> file "ietf-netconf-ex@2014-04-21.yang"

```
module ietf-netconf-ex {  
  
    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-ex";  
    prefix ncex;  
  
    import ietf-inet-types {  
        prefix inet;  
    }  
  
    import ietf-netconf {  
        prefix nc;  
    }  
  
    import ietf-netconf-notifications {  
        prefix ncn;  
    }  
  
    import ietf-netconf-with-defaults {  
        prefix ncwd;  
    }  
  
    import ietf-yang-patch {  
        prefix yp;  
    }  
  
    import ietf-yang-types {  
        prefix yang;  
    }  
  
    organization
```



```

    "IETF NETCONF (Network Configuration Protocol) 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: Bert Wijnen
                <mailto:bertietf@bwijnen.net>

    Editor:    Andy Bierman
                <mailto:andy@yumaworks.com>";

description
    "This module contains a collection of YANG definitions for the
    efficient operation of a NETCONF server. Protocol operations
    are defined to reduce network usage and transaction complexity.

    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.";

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

// RFC Ed.: remove this note
// Note: extracted from
// draft-bierman-netconf-efficiency-extensions-01.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision "2014-04-21" {
    description
        "Initial revision. <get2> operation originally published
        in draft-bierman-netconf-get2-03.txt";
    reference
        "RFC XXXX: NETCONF Efficiency Extensions";

```


Internet-Draft

NETCONF-EX

April 2014

```
}
```

```
/* Features */
```

```
feature timestamps {  
  description
```

```
    "This feature indicates that the server implements  
    the <get2> operations parameters which require  
    last modification timestamps to be maintained by  
    the server.
```

```
  
    If this feature is advertised then one global  
    'last-modified' timestamp for the entire  
    running configuration datastore MUST be supported.
```

```
  
    The server MAY support additional timestamps  
    for additional datastores and data nodes  
    within a datastore. The 'with-metadata'  
    parameter can be used to identify  
    which data nodes support a 'last-modified'  
    timestamp.";
```

```
}
```

```
feature with-defaults {  
  description
```

```
    "This feature indicates that the server supports the  
    'with-defaults' parameter for the <get2> operation.  
    A NETCONF server SHOULD support this feature.";
```

```
  reference
```

```
    "RFC 6243: With-defaults Capability for NETCONF";
```

```
}
```

```
feature confirmed-edit {  
  description
```

```
    "This feature indicates that the server supports the  
    confirmed commit procedure for the <edit2> protocol  
    operation.";
```

```
}
```

```
/* Identities */
```



```
identity metadata {
  description
    "Base for all metadata identifiers used by the
    'with-metadata' parameter in the <get2> operation.";
}
```

```
identity timestamps {
  base metadata;
  description
    "Describes metadata identifying the last modification
    time of the associated datastore or data resource.";
}
```

```
identity etags {
  base metadata;
  description
    "Describes metadata identifying the entity tag value
    of the associated datastore or data resource.";
}
```

```
identity config-id {
  base metadata;
  description
    "Describes metadata identifying the config ID
    of the associated datastore or data resource.";
}
```

```
/* Typedefs */
```

```
typedef yang-entity-tag {
  type string;
  description
    "Contains an opaque string representing a specific instance
    of a datastore or data resource. A client can use this
    string for equality comparisons between yang-entity-tag
    values.
```

If any configuration data node values changes, or the relative order of any user-ordered data changes, then the server MUST change the entity tag value for the running datastore to a different value. If the server

maintains entity-tag values for configuration data nodes, then the server MUST change the yang-entity-tag value for any affected data node.

Only yang-entity-tag values for the same target resource instance can be compared. Only the 'strong entity tag' form is required. A server MAY support the 'weak entity tag' form. If so, then 2 YANG data node resource instances are considered to be equivalent if they contain the same value subtrees and all user-ordered data nodes share the same relative order.";

reference

```
    "RFC 2616, section 3.11.";
}

/* Groupings */

grouping lock-parms {
  description
    "Common parameters to control datastore locking.";

  leaf with-locking {
    type empty;
    description
      "If this parameter is present then the request MUST be
       performed with exclusive write access to all datastores
       involved in the operation. An 'operation-not-supported'
       error-tag value is returned if the target datastore for
       the operation does not support locking (e.g., 'url' or
       'operational')."

      If the server cannot provide exclusive write access
      for the entire requested operation then an 'in-use'
      error-tag value is returned.

      If the 'max-lock-wait' parameter is also present then
      the server MAY choose to wait up to that amount of
      time attempting to obtain exclusive write access,
      before returning an error.";
  }
}
```



```

leaf max-lock-wait {
  when "../with-locking" {
    description
      "Only relevant if locking is requested.";
  }
  type uint32 {
    range "1 .. 600";
  }
  units seconds;
  description
    "If this parameter is present and the 'with-locking'
    parameter is also present, then the server MAY wait
    up to the specified number of seconds attempting
    to obtain exclusive write access for the requested
    operation.";
}
}

```

```

/* Protocol Operations */

rpc get2 {
  description
    "Retrieve NETCONF datastore information";
  input {
    container source {
      description
        "The datastore (or non-configuration data)
        to use for the source for the retrieval operation.";

      choice datastore-source {
        default running;
        description
          "The configuration source for the retrieval operation.
          The running configuration is the default choice if
          this parameter is not present.";
        leaf candidate {
          if-feature nc:candidate;
          type empty;
          description
            "The candidate configuration datastore is the

```



```

        retrieval source.";
    }
    leaf running {
        type empty;
        description
            "The running configuration datastore is the
            retrieval source.";
    }
    leaf startup {
        if-feature nc:startup;
        type empty;
        description
            "The startup configuration datastore is the
            retrieval source.";
    }
    leaf url {
        if-feature nc:url;
        type inet:uri;
        description
            "The URL-based configuration is the
            retrieval source.";
    }
    leaf operational {
        type empty;
        description
            "The retrieval source is the collection of all

```

operational (non-configuration) data nodes supported by the server.

Any ancestor container and/or list and list key nodes are also returned. No other leafs or leaf-lists will be included in the reply.

The server MAY return ancestor container, and/or list and list key nodes that do not contain any non-configuration nodes. This can occur for several reasons, e.g., the implementation streams replies and cannot defer instrumentation or access control filtering of descendant data nodes.";

```

    }
}

```



```

}

choice filter-spec {
  description
    "The content filter specification for this request";

  anyxml subtree-filter {
    description
      "This parameter identifies the portions of the
        target datastore to retrieve.";
    reference "RFC 6241, Section 6.";
  }
  leaf xpath-filter {
    if-feature nc:xpath;
    type yang:xpath1.0;
    description
      "This parameter contains an XPath expression
        identifying the portions of the target
        datastore to retrieve.";
  }
}

leaf keys-only {
  type empty;
  description
    "This parameter selects only data nodes which
      are key leaf nodes.  Parent container and
      list nodes are also returned, but no other leafs,
      or any leaf-lists will be included in the reply.";
}

leaf if-modified-since {
  if-feature timestamps;

```

```

type yang:date-and-time;
description
  "This parameter selects the target datastore
    only if the last-modified timestamp for the
    datastore is more recent than the specified time.
    If not, then an empty <data> element is returned.

    If the target datastore does not maintain a

```



```

        last-modified timestamp, then this parameter is
        ignored.";
    }

    leaf full-delta {
        if-feature timestamps;
        type empty;
        description
            "This parameter selects only data nodes which
            have been modified since the specified time.
            It is ignored unless the 'if-modified-since'
            parameter is also provided and the target datastore
            supports a last-modified timestamp.";
    }

    leaf depth {
        type uint32;
        default 0;
        description
            "This parameter selects how many conceptual
            sub-tree levels should be returned in the
            <rpc-reply>.

            If this parameter is equal to '0', then entire
            subtrees will be returned.

            If this parameter is greater than '0', then
            only the specified number of subtree levels will
            be returned.";
    }

    uses ncwd:with-defaults-parameters {
        if-feature with-defaults;
        description
            "This parameter controls the retrieval of
            default values.";
        reference
            "RFC 6243: With-defaults Capability for NETCONF";
    }

```

```

leaf-list with-metadata {

```



```

    type identityref {
        base metadata;
    }
    description
        "This parameter will cause the server to return
        metadata in the <rpc-reply> (e.g. as XML attributes
        in XML encoding) associated with the specified
        metadata identity. If the server does not support
        any specified metadata identifier, then the
        operation fails with an 'invalid-value' error.";
}

uses lock-parms {
    description
        "Exclusive write access can be requested to
        ensure that no other sessions modify the
        configuration data during the retrieval operation";
}

}

output {
    anyxml data {
        description
            "Copy of the requested datastore subset which
            matched the filter criteria (if any).
            An empty data container indicates that the
            request did not produce any results.";
    }
}

}

rpc edit2 {
    description
        "Edit NETCONF datastore contents.
        All operations requested in the yang-patch edit list
        are applied, or the target datastore is left unchanged.";

    input {
        container target {
            description
                "The datastore to use as the target for this
                edit operation.";

            choice datastore-target {
                mandatory true;
            }
        }
    }
}

```



```
description
  "The configuration target for the edit operation.";

leaf candidate {
  if-feature nc:candidate;
  type empty;
  description
    "The candidate configuration datastore is the
    edit target.";
}
leaf running {
  if-feature nc:writable-running;
  type empty;
  description
    "The running configuration datastore is the
    edit target.";
}
}
```

```
leaf target-resource {
  if-feature nc:xpath;
  type yang:xpath1.0;
  description
    "This parameter identifies 1 or more data node
    instances for which the yang-patch edits
    will be applied. The target-resource expression
    MUST evaluate to a node-set result.
```

Each operation in the yang-patch edit list will be applied to each target-resource instance, as if it were the document root for the operation.

If multiple instances are represented by the target-resource value, then the server will apply all edits to all instances. If any errors occur, then all edits from this request will be undone from the target datastore.

The user **MUST** have appropriate write permissions for all data accessed by every operation within the edit list.

If this parameter is not present or not supported then the target resource is the root node of the datastore identified by the 'target' parameter.";

}

```
uses yp:yang-patch {
  description
    "The yang-patch parameter contains the ordered list
    of edits to perform on the target resource(s).

    The conceptual document root for the 'target'
    parameter is defined to be the value of a data node
    represented by the 'target-resource' parameter or the
    target datastore conceptual root node if that parameter
    is not present.";
```

```
}
```

```
leaf test-only {
  type empty;
  description
    "If this parameter is present the server will not
    actually perform the requested edits. Instead the
    edit request will be validated as if it were going
    to be applied. Any parameter errors or datastore
    validation errors SHOULD be reported in the response.

    No attempt to apply, activate the edits or save them
    in non-volatile storage will be made if this parameter
    is present.";
```

```
}
```

```
leaf if-match {
  type yang-entity-tag;
  description
    "If this parameter is set, then the entire edit request
    will be rejected unless the entity tag for the target
    resource matches this value. An rpc-error with
    an 'operation-failed' error-tag value MUST be returned,
    and the edit operation MUST NOT be attempted.
    The 'error-app-tag' field SHOULD be set to
    'precondition-failed'.

    If the target datastore does not maintain a
    last-modified timestamp, then this parameter is
```



```

        ignored.";
    }

    uses lock-parms {
        description
            "Exclusive write access can be requested to
            ensure that no other sessions modify the
            configuration data during the edit operation
            and possibly the entire confirmed commit procedure.

```

Bierman

Expires October 23, 2014

[Page 39]

Internet-Draft

NETCONF-EX

April 2014

If the 'with-locking' parameter is used to start or extend a confirmed commit procedure, then the exclusive write access will be maintained until the confirmed commit procedure terminates somehow.

If the 'with-locking' parameter is used for a plain edit operation, then exclusive write access will be maintained until this operation has completed.";

```

    }

    leaf activate-now {
        type empty;
        description
            "If present and the edit operation succeeds,
            then the server will activate the configuration
            changes right away. The server will conceptually
            perform a <commit> operation after the edit
            operation. The user MUST have execute
            permission for the <commit> operation or
            the operation fails with an 'access-denied' error.

            This parameter has no affect unless the
            'datasource-target' choice is the 'candidate' leaf.";
    }

```

```

    leaf nvstore-now {
        type empty;
        description
            "If present and the edit operation succeeds,
            and the configuration changes are activated
            in the running datastore, then the server
            will persist the configuration changes right away

```


in non-volatile store. The server will conceptually perform a <copy-config> operation from the running to the startup datastore. The user MUST have execute permission for the <copy-config> operation or the operation fails with an 'access-denied' error.

This parameter has no affect unless the
'startup' capability is supported by the server.";
}

```
leaf confirmed {  
  if-feature confirmed-edit;  
  type empty;  
  description  
    "If the requested edit operation succeeds and the  
    configuration changes are applied to the running
```

datastore, then a confirmed commit procedure is requested by the client.

A confirmed commit procedure is an <edit2> operation that contains this parameter. The <complete-commit> operation is used to complete the confirmed commit procedure. The <revert-commit> operation is used to cancel the confirmed commit procedure and revert the running datastore back to the contents before the first confirmed commit operation.

If no <complete-commit> or <revert-commit> operation is invoked within the timeout interval then the server will revert the running datastore back to the contents before the first confirmed edit operation.

This is the same as the confirmed commit procedure in [RFC 6241](#) except the candidate capability is not required.

The server will save the running datastore contents before the edit operation is activated, if there is no confirmed edit already in progress.


```

        If the 'with-locking' parameter is present then
        the server will maintain exclusive write access
        for the specified session until the confirmed
        edit procedure is completed somehow.";
    reference
        "RFC 6241, Section 8.3.4.1";
}

leaf confirm-timeout {
    when "../confirmed" {
        description
            "Only relevant if the <confirmed>parameter is present";
    }
    if-feature confirmed-edit;
    type uint32 {
        range "1..max";
    }
    units "seconds";
    default "600";    // 10 minutes
    description
        "The timeout interval for a confirmed edit procedure.
        If exclusive write access was granted for this confirmed
        commit procedure, then it is removed if the timeout

```

```

        occurs and the confirmed commit procedure is terminated.";
    reference "RFC 6241, Section 8.3.4.1";
}

leaf persist {
    if-feature confirmed-edit;
    type string;
    description
        "This parameter is used to make a confirmed commit
        procedure persistent. A persistent confirmed commit
        is not aborted if the NETCONF session terminates.
        The only way to abort a persistent confirmed commit
        is to let the timer expire, or to use the
        <revert-commit> operation.

        The value of this parameter is a token that MUST be
        given in the 'persist-id' parameter of the <edit2>,

```


<complete-commit>, or <revert-commit> operations in order to extend, confirm, or cancel the persistent confirmed commit procedure.

```
    The token SHOULD be a random string.";
    reference "RFC 6241, Section 8.3.4.1";
}

leaf persist-id {
    if-feature confirmed-edit;
    type string;
    description
        "This parameter is given in order to extend a persistent
        confirmed edit. The value must be equal to the value
        given in the 'persist' parameter to the <commit>
        operation. If it does not match, the operation fails
        with an 'invalid-value' error.";
    reference "RFC 6241, Section 8.3.4.1";
}
}

output {
    uses yp:yang-patch-status;
}
}

rpc complete-commit {
    if-feature confirmed-edit;
    description
```

"This operation is used to complete an ongoing confirmed commit procedure. If exclusive write access was granted for this confirmed commit procedure, then it is removed if this operation is successfully completed.

If the confirmed commit is persistent, the parameter 'persist-id' MUST be given, and it MUST match the value of the 'persist' parameter given in the <edit2> operation. If not confirmed commit procedure is in progress then the operation fails with an 'operation-failed' error."; reference "[RFC 6241, Section 8.4.5.1](#)";


```

input {
  leaf persist-id {
    type string;
    description
      "This parameter is given in order to complete a
      persistent confirmed commit procedure. The
      value MUST be equal to the value given in the
      'persist' parameter to the <edit2> operation.
      If it does not match, the operation fails with
      an 'invalid-value' error.";
  }
}
}

rpc revert-commit {
  if-feature confirmed-edit;
  description
    "This operation is used to cancel an ongoing confirmed commit.
    If exclusive write access was granted for this confirmed
    commit procedure, then it is removed if this operation
    is successfully completed.

    If the confirmed commit is persistent, the parameter
    'persist-id' MUST be given, and it MUST match the value of
    the 'persist' parameter. If not confirmed commit
    procedure is in progress then the operation fails
    with an 'operation-failed' error.";
  reference "RFC 6241, Section 8.4.4.1";

  input {
    leaf persist-id {
      type string;
      description
        "This parameter is given in order to cancel a persistent
        confirmed commit and revert the running configuration

```

datastore to its state before the confirmed commit procedure started. The value MUST be equal to the value given in the 'persist' parameter to the <edit2> operation.


```

        If it does not match, the operation fails with an
        'invalid-value' error.";
    }
}

/* Notifications */

augment /ncn:netconf-config-change {
    description
        "Add the updated config-id capability value
        to a configuration change event.";

    leaf config-id {
        type string;
        description
            "Contains the new configuration ID for the
            running datastore on the server, representing
            the datastore after the configuration changes.";
    }
}

}

<CODE ENDS>

```

2.8. XSD for NETCONF-EX Metadata

The following XML Schema document [[XSD](#)] defines the "last-modified" and "etag" attributes, described within this document. The "last-modified" attribute is only relevant if the server supports the "timestamps" YANG feature within the "ietf-netconf-ex" YANG module.

The "last-modified" attribute uses the XSD data type "dateTime", in accordance with [Section 3.2.7.1](#) of XML Schema Part 2: Datatypes. This is equivalent to the YANG data type "date-and-time".

The "etag" attribute uses the XSD data type "string", in accordance with the "yang-entity-tag" YANG typedef defined in [Section 2.7](#).

The "config-id" attribute uses the XSD data type "string".

```
<CODE BEGINS> file "netconf-ex.xsd"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0"
  targetNamespace="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xml:lang="en">

  <xs:annotation>
    <xs:documentation>
      This schema defines the syntax for the "last-modified"
      and "etag" attributes described within this document.
    </xs:documentation>
  </xs:annotation>

  <!--
    config-id attribute
  -->
  <xs:attribute name="config-id" type="xs:string">
    <xs:annotation>
      <xs:documentation>
        This attribute indicates the current config ID
        for the running configuration datastore,
        corresponding to the XML element containing this attribute.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <!--
    last-modified attribute
  -->
  <xs:attribute name="last-modified" type="xs:dateTime">
    <xs:annotation>
      <xs:documentation>
        This attribute indicates the date and time when
        a modification was last detected by the server
        for the datastore or data node corresponding to
        the XML element containing this attribute.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <!--
```


etag attribute

Bierman

Expires October 23, 2014

[Page 45]

Internet-Draft

NETCONF-EX

April 2014

```
-->
<xs:attribute name="etag" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      This attribute indicates the entity tag
      for the datastore or data node corresponding to
      the XML element containing this attribute.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

</xs:schema>

<CODE ENDS>
```


[3.](#) IANA Considerations

[3.1.](#) NETCONF-EX XML Namespace

This document registers a URI in the IETF XML registry [[RFC3688](#)]. Following the format in [RFC 3688](#), the following registration is requested:

```
URI: urn:ietf:params:xml:ns:netconf:netconf-ex:1.0
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

[3.2.](#) NETCONF-EX XML Schema

This document registers a URI for the NETCONF XML schema in the IETF XML registry [[RFC3688](#)].

```
// RFC Ed. remove this line and uncomment next line when published
//IANA has updated the following URI to reference this document.
```

```
URI: urn:ietf:params:xml:schema:netconf-ex
```

[3.3.](#) NETCONF-EX YANG Module

This document registers 1 YANG module in the YANG Module Names registry [[RFC6020](#)].

```
name:          ietf-netconf-ex
namespace:     urn:ietf:params:xml:ns:yang:ietf-netconf-ex
prefix:        ncex
// RFC Ed. remove this line and replace XXXX in next line
reference:     RFC XXXX
```


[4.](#) Security Considerations

This document does not introduce any new security concerns in addition to those specified in [\[RFC6241\], section 9](#).

[5.](#) Change Log

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

[5.1.](#) 00 to 01

[5.1.1.](#) Removed :capability-id Capability

The :capability-id URI exchange was removed because the NETCONF protocol does not allow the server to delay its <hello> message so the client cannot choose the full or abbreviated <hello>.

This makes the :capability-id URI exchange unworkable for several reasons:

- o Since the client cannot select the server hello format based on its own notion of the cached capability set, the server must be configured to always use the full or always use the abbreviated <hello> message.

- o All clients must support the new capability exchange or the server cannot practically be configured to use the abbreviated <hello> message.
- o Since the client will not know the capability-id value for a server the first time the particular value is seen, the "schema" list in the "ietf-netconf-monitoring" YANG module would have to be mandatory-to-implement by both client and server, and mandatory-to-use by the client.
- o Forcing the client to perform a <get> request and wait for an <rpc-reply> before using the NETCONF session introduces 1 round-trip of extra latency into the protocol.
- o Forcing the client to perform a <get> request and wait for an <rpc-reply> before using the NETCONF session introduces extra complexity into the protocol.

[5.1.2.](#) RESTCONF Alignment

The YANG module was updated to align with new RESTCONF and YANG Patch drafts. The "location" leaf has been removed from the "yang-patch-status" grouping.

[6.](#) Normative References

[I-D.lhotka-netmod-json]

Lhotka, L., "Modeling JSON Text with YANG",
[draft-lhotka-netmod-yang-json-02](#) (work in progress),
 September 2013.

[JSON]

Bray, T., Ed., "The JSON Data Interchange Format",
[draft-ietf-json-rfc4627bis-03](#) (work in progress),
 September 2013.

[RESTCONF]

Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "RESTCONF Protocol", [draft-ietf-netconf-restconf-00](#) (work in progress), March 2014.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), January 2004.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", [RFC 6020](#), October 2010.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", [RFC 6241](#), June 2011.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", [RFC 6243](#), June 2011.
- [RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", [RFC 6470](#), February 2012.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", [RFC 6991](#), July 2013.
- [XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.
- [XSD] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.

[YANG-Patch]

Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "YANG Patch Media Type", [draft-ietf-netconf-yang-patch-00](#) (work in progress), March 2014.

[Appendix A](#). Open Issues

[A.1](#). resource-identifier-type

The resource-identifier-type typedef from yang-patch is a RESTCONF path expression, not an XPath path expression. The error-path parameter also uses RESTCONF path strings. Should either or both of these be XPath instead?

[A.2](#). no YANG for top-level message nodes

The YANG module of the node is needed for JSON encoding, but there is no YANG schema definition for the <rpc>, <rpc-reply>, or <notification> elements. The namespace for <rpc> and <rpc-reply> is "ietf-netconf", but no module name at all exists for the <notification> element.

[A.3](#). config-id attribute

Should the "config-id" (etag for the running datastore root) be returned in every <get2> response or only if requested? (Currently only if requested.)

[A.4](#). <get2> nodeset retrieval

Should there be a retrieval mode for <get2> where only the nodes in an XPath node-set are returned? NETCONF returns all ancestor nodes and all ancestor or sibling key leafs as well. Sometime the XPath designer knows the context of the result node-set (e.g. path expression for 1 instance of a nested list). The XML scaffolding can add a lot of extra bytes to the <rpc-reply>.

[Appendix B](#). Additional Examples

[B.1](#). YANG Module Used in Examples

The "example-ex" YANG module models a collection of forests. Each forest has a collection of trees. For simplicity, only 1 tree of each type is allowed in a forest.

```

+--rw forests
  +--rw forest [name]
    +--rw name          string
    +--ro tree-count?   uint32
    +--rw trees
      +--rw tree [name]
        +--rw name      string
        +--rw location? string
        +--ro height?   decimal64

module example-ex {

  namespace "http://example.com/ns/example-ex";
  prefix ex;
  organization "Example, Inc.";
  contact "support@example.com";

  description "Module used in NETCONF-EX examples.";
  revision 2013-10-19 {
    description "Initial version";
    reference "Example Spec 12.44";
  }

  container forests {
    description "A collection of forests";

    list forest {
      key name;
      description "A single forest";

      leaf name {
        type string;
        description "The forest name";
      }
    }
  }
}
```



```

}

leaf tree-count {
    type uint32;
    config false;
    description "The number of trees in this forest";
}

```

```

}

container trees {
    description "A collection of trees";

    list tree {
        key name;
        description "A single tree";

        leaf name {
            type string;
            description "The tree name";
        }
        leaf location {
            type string;
            description "The tree location";
        }

        leaf height {
            type decimal64 {
                fraction-digits 3;
            }
            units meters;
            config false;
            description "The tree height";
        }
    } // list tree
} // container trees
} // list forest
} // container forests
}

```

[B.2.](#) YANG Data Used in Examples

The follow instances are assumed in the following examples.


```
list forest: "north":  
  list tree: "birch", "ash", "maple"
```

```
list forest: "south":  
  list tree: "banyan", "palm"
```

```
leaf "location": "hillside", "west valley", "southwest pasture",  
  "east meadow", "greenhouse"
```

The forests and trees are configured, which represent trees the company has planted and growing over time.

The operational data (tree height) represents the data that the company monitors for each tree over time.

[B.3.](#) <edit2> Examples

[B.3.1.](#) Confirmed Commit on the "running" Datastore

In this example, the server supports the :writable-running and :startup capabilities:

```
<rpc message-id="105"  
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"  
    xmlns:ex="http://example.com/ns/example-ex">  
    <target><running/></target>  
    <target-resource>  
      /ex:forests/ex:forest[ex:name='north']  
    </target-resource>  
    <yang-patch>  
      <patch-id>oak-tree-patch</patch-id>  
      <comment>Create an oak tree</comment>  
      <edit>  
        <edit-id>oak</edit-id>  
        <operation>create</operation>  
        <target>/ex:trees</target>  
        <value>  
          <ex:tree>  
            <ex:name>oak</ex:name>
```



```

        <ex:location>hillside</ex:location>
      </ex:tree>
    </value>
  </edit>
</yang-patch>
<with-locking/>
<max-lock-wait>10</max-lock-wait>
<confirmed/>
<confirm-timeout>60</confirm-timeout>
<persist>24ef8829a4</persist>
</edit2>
</rpc>

```

The edit succeeds, and the "yang-patch-status" container is returned to the client with the <location> path expression of the new oak tree resource. The candidate and running datastores remain locked after this operation because a confirmed commit procedure is in progress. The startup datastore was not locked during this operation because the "nvstore-now" parameter was not provided.

```

<rpc-reply message-id="105"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>oak-tree-patch</patch-id>
    <ok/>
    <edit-status>
      <edit>
        <edit-id>oak</edit-id>
        <ok/>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>

```

After configuration verification (e.g., 20 seconds), the client decides to keep these configuration changes and sends a <complete-commit> request.

```

<rpc message-id="106"

```



```

    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <complete-commit
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
      <persist>24ef8829a4</persist>
    </complete-commit>
  </rpc>

```

The server completes the confirmed commit procedure and returns an "ok" element to indicate success:

```

<rpc-reply message-id="106"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

```

After the operation succeeds, the server releases all locks that were being held to allow exclusive write access for the entire confirmed commit procedure.

The client can now save the activated configuration changes to the startup configuration using the <copy-config> protocol operation, as described in [RFC 6241, section 8.7.5.1](#).

[B.3.2](#). Conditional Editing with "if-match" Parameter

In this example the client is going to change the location of the "palm" tree is the "south" forest. The entity tag for the tree

resource is retrieved with the resource:

```

<rpc message-id="107"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <xpath-filter> <!-- wrapped for display -->
      /ex:forests/ex:forest[ex:name='south']/ex:trees/
      ex:tree[ex:name='palm']
    </xpath-filter>
    <depth>1</depth>
    <with-metadata>ncex:etags</with-metadata>
  </get2>
</rpc>

```


The server returns a subtree containing data nodes representing the "palm" tree. The "etag" attribute is returned for this resource and its ancestors. Only the "tree" node itself, as requested with the "depth" parameter.

```
<rpc-reply message-id="107"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    lm:last-modified="2012-09-09T02:00:00Z">
    <forests xmlns="http://example.com/ns/example-ex"
      lm:etag="34ef6892">
      <forest lm:etag="ef11eb99">
        <name>south</name>
        <trees lm:etag="ef11eb99">
          <tree lm:etag="3477cc82" />
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

The client then edits the list entry (e.g, reassigns tree location) but submits an "if-match" parameter with the "etag" value it received for the tree resource being edited:

```
<rpc message-id="108"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource> <!-- wrapped for display -->
      /ex:forests/ex:forest[ex:name='south']/ex:trees
```



```

    /ex:tree[ex:name='palm']
  </target-resource>
  <yang-patch>
    <patch-id>move-palm-tree</patch-id>
    <comment>Move the palm tree</comment>
    <edit>
      <edit-id>palm</edit-id>
      <operation>merge</operation>
      <target></target>
      <value>
        <ex:location>greenhouse</ex:location>
      </value>
    </edit>
  </yang-patch>
  <if-match>3477cc82</if-match>
  <with-locking/>
  <max-lock-wait>10</max-lock-wait>
  <activate-now/>
  <nvstore-now/>
</edit2>
</rpc>

```

In this example the tree resource has been edited by another client since the <get2> reply for this client, so the edit request is not even attempted. Instead an "operation-failed" is returned:


```

<rpc-reply message-id="108"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>move-palm-tree</patch-id>
    <errors>
      <error>
        <error-type>protocol</error-type>
        <error-tag>operation-failed</error-tag>
        <error-app-tag>precondition-failed</error-app-tag>
        <error-path> <!-- wrapped for display -->
          /ex:forests/ex:forest[ex:name='south']/ex:trees/
          ex:tree[ex:name='palm']
        </error-path>
        <error-message xml:lang="en">
          if-match precondition failed
        </error-message>
      </error>
    </errors>
  </yang-patch-status>
</rpc-reply>

```

[B.3.3.](#) Bulk Editing with "target-resource" Parameter

In this example, the server supports the :candidate and :startup capabilities, so all 3 datastores (including running) are locked for the <edit2> operation. There is a new pine tree for each forest that is being created and sent to the greenhouse.

Internet-Draft

NETCONF-EX

April 2014

```
<rpc message-id="109"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource>
      /ex:forests/ex:forest
    </target-resource>
    <yang-patch>
      <patch-id>pine-tree-patch</patch-id>
      <comment>Add 2 new pine trees to greenhouse</comment>
      <edit>
        <edit-id>pine</edit-id>
        <operation>create</operation>
        <target>/ex:trees</target>
        <value>
          <ex:tree>
            <ex:name>pine</ex:name>
            <ex:location>greenhouse</ex:location>
          </ex:tree>
        </value>
      </edit>
    </yang-patch>
    <with-locking/>
    <max-lock-wait>10</max-lock-wait>
    <activate-now/>
    <nvstore-now/>
  </edit2>
</rpc>
```

The edit succeeds, and the "yang-patch-status" container is returned to the client with the status information.

```
<rpc-reply message-id="109"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>pine-tree-patch</patch-id>
    <ok/>
    <edit-status>
      <edit>
        <edit-id>pine</edit-id>
```



```
        <ok/>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>
```

[B.3.4.](#) Edit Validation with "test-only" Parameter

In this example, the client is checking if it can change the location field in the "palm" tree list entry by using the "test-only" parameter:

```
<rpc message-id="110"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource>
      /ex:forests/ex:forest[ex:name='south']/ex:trees
    </target-resource>
    <yang-patch>
      <patch-id>palm-tree-move</patch-id>
      <comment>Move the palm tree to riverside</comment>
      <edit>
        <edit-id>palm</edit-id>
        <operation>merge</operation>
        <target>/ex:tree/palm</target>
        <value>
          <ex:location>riverside</ex:location>
        </value>
      </edit>
    </yang-patch>
    <test-only/>
  </edit2>
</rpc>
```

Since "riverside" is not a supported location, an "invalid-value" error is returned for the requested edit operation:


```
<rpc-reply message-id="110"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>palm-tree-move</patch-id>
    <edit-status>
      <edit>
        <edit-id>palm</edit-id>
        <errors>
          <error>
            <error-type>protocol</error-type>
            <error-tag>invalid-value</error-tag>
            <error-path> <!-- wrapped for display -->
              /ex:forests/ex:forest[ex:name='south']/ex:trees/
              ex:tree[ex:name='palm']
            </error-path>
            <error-message xml:lang="en">
              value is invalid
            </error-message>
          </error>
        </errors>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>
```

[B.4.](#) <get2> Examples

B.4.1. If-Modified-Since Non-Empty Filter Retrieval

In this example, the running datastore was last modified at "2012-09-09T01:43:27Z" because the forest named "north" was modified at this time.

- o The forest named "north" was last modified after the specified "if-modified-since" timestamp.
- o The forest named "south" was last modified before the specified "if-modified-since" timestamp.
- o The server maintains a last-modified timestamp for the running datastore and the "forest" list entries.
- o The client is requesting only the changed entries after 2012-09-09T01:43:27Z, so the "full-delta" parameter is set.

- o The client is also requesting that timestamps be returned within the data nodes. If any part of the "forest" subtree is modified then this timestamp will be updated.

```
<rpc message-id="111"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
    <if-modified-since>2012-09-09T01:43:27Z</if-modified-since>
    <full-delta/>
    <with-metadata>ncex:timestamps</with-metadata>
  </get2>
</rpc>

<rpc-reply message-id="111"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
```



```

lm:last-modified="2012-09-09T02:00:00Z">
  <forests xmlns="http://example.com/ns/example-ex">
    <forest lm:last-modified="2012-09-09T02:00:00Z">
      <name>north</name>
      <trees>
        <tree>
          <name>birch</name>
          <location>hillside</location>
        </tree>
        <tree>
          <name>ash</name>
          <location>southwest pasture</location>
        </tree>
        <tree>
          <name>maple</name>
          <location>east meadow</location>
        </tree>
      </trees>
    </forest>
  </forests>
</data>
</rpc-reply>

```

[B.4.2.](#) If-Modified-Since Empty Filter Retrieval

In this example the client has changed the "if-modified-since" timestamp to a time in the future.

- o No "forest" list entry has been modified since this time so an empty data node is returned.
- o Note that the "last-modified" timestamp is returned for the node representing the datastore, even though no data nodes have been modified since the specified time. This allows the client to easily retrieve the last-modified timestamp for the entire datastore.


```

<rpc message-id="112"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
    <if-modified-since>2012-09-09T03:43:27Z</if-modified-since>
    <with-metadata>ncex:timestamps</with-metadata>
  </get2>
</rpc>

<rpc-reply message-id="112"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    lm:last-modified="2012-09-09T02:00:00Z" />
</rpc-reply>

```

[B.4.3.](#) Keys Only Filter Retrieval

This example retrieves only the names from the "forests" subtree in the running datastore.

- o The default source (running) is used.
- o The default depth="0" is used to retrieve all subtree levels.
- o The "keys-only" leaf is set
- o The "forests" subtree is selected. The xpath-filter is used instead of the subtree-filter.

- o Whitespace added to xpath-filter element for display purposes only.

```

<rpc message-id="113"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <xpath-filter xmlns:ex=http://example.com/ns/example-ex">

```



```

    /ex:forests
  </xpath-filter>
  <keys-only />
</get2>
</rpc>

<rpc-reply message-id="113"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest>
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
          </tree>
          <tree>
            <name>ash</name>
          </tree>
          <tree>
            <name>maple</name>
          </tree>
        </trees>
      </forest>
      <forest>
        <name>south</name>
        <trees>
          <tree>
            <name>banyan</name>
          </tree>
          <tree>
            <name>palm</name>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>

```


This example retrieves the "trees" node to determine which forests have any trees.

- o Only 1 subtree level is requested, instead of the default of all levels.
- o The default source (running) is used.
- o The "trees" subtree is selected.
- o The depth parameter is set to "1" to only retrieve the requested layer "trees" and its ancestor nodes and the configuration leaf nodes from each "forest" entry.

```
<rpc message-id="114"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex">
        <forest>
          <trees />
        </forest>
      </forests>
    </subtree-filter>
    <depth>1</depth>
  </get2>
</rpc>
```

```
<rpc-reply message-id="114"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest>
        <name>north</name>
        <trees />
      </forest>
      <forest>
        <name>south</name>
        <trees />
      </forest>
    </forests>
  </data>
</rpc-reply>
```


[B.4.5.](#) Retrieve Only Non-Configuration Data Nodes

This example retrieves only the name leafs from the "forest" list within the "forests" subtree, in the running datastore.

- o The "source" leaf is set to the "operational" data source
- o The "forests" subtree is selected

Internet-Draft

NETCONF-EX

April 2014

```
<rpc message-id="115"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <source><operational/></source>
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
  </get2>
</rpc>
```

```
<rpc-reply message-id="115"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest>
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
            <height>41.013</height>
          </tree>
          <tree>
            <name>ash</name>
            <height>16.523</height>
          </tree>
          <tree>
            <name>maple</name>
            <height>51.204</height>
          </tree>
        </trees>
      </forest>
      <forest>
        <name>south</name>
        <trees>
          <tree>
            <name>banyan</name>
            <height>91.433</height>
          </tree>
          <tree>
```



```
        <name>palm</name>
        <height>83.439</height>
    </tree>
</trees>
</forest>
</forests>
</data>
</rpc-reply>
```

Bierman

Expires October 23, 2014

[Page 68]

Internet-Draft

NETCONF-EX

April 2014

Author's Address

Andy Bierman
YumaWorks, Inc.

Email: andy@yumaworks.com

Bierman

Expires October 23, 2014

[Page 69]