

# **Initial YANG-next Impact on NETCONF 2.0**

**Kent Watsen  
NETCONF WG  
IETF 125**

# Introduction

The YANG-next "baseline" removes NETCONF-specific bits from RFC 7950.

- see diffs [here](#)

These bits should move to the "NETCONF 2.0" document including

# Sections Removed from RFC 7950

- Section 5.1.2.1 (NETCONF XML Encoding)
- Section 5.3.1 (YANG XML Namespace)
- Section 5.6.4 (Announcing Conformance Information in NETCONF)
- Section 7.5.8 (NETCONF <edit-config> Operations, for containers)
- Section 7.6.7 (NETCONF <edit-config> Operations, for leafs)
- Section 7.7.9 (NETCONF <edit-config> Operations, for leaf-lists) <-- Surprise!
- Section 7.7.10 (Usage Example) for leaf-lists
- Section 7.8.6 (NETCONF <edit-config> Operations, for lists) <-- Surprise!
- Section 7.8.7 (Usage Example) for lists
- Section 7.10.3 (NETCONF <edit-config> Operations, for anydata)
- Section 7.11.3 (NETCONF <edit-config> Operations, for anyxml)
- Section 7.14.4 (NETCONF XML Encoding Rules, for rpc)
- Section 7.14.5 (Usage Example, for rpc)
- Section 7.15.2 (NETCONF XML Encoding Rules, for action)
- Section 7.15.3 (Usage Example, for action)
- Section 7.16.2 (NETCONF XML Encoding Rules, for notification)
- Section 7.16.3 (Usage Example, for notification)

# Biggest Surprise

Nowhere in RFC 6241 are the "insert" and "key" attributes defined

- for controlling "ordered-by user" lists and leaf-lists.

History

- NETCONF (RFC 4741) was published before YANG 1.0 (RFC 6020)
  - thus ordered lists didn't exist for RFC 4741

But NETCONF was republished in RFC 6241

- that is, *after* RFC 6020
  - and so **should** have defined how to control ordering, right? 🙄

# Details

# Section 5.1.2.1 (NETCONF XML Encoding)

## 5.1.2.1. NETCONF XML Encoding

NETCONF is capable of carrying any XML content as the payload in the <config> and <data> elements. The top-level nodes of YANG modules are encoded as child elements, in any order, within these elements. This encapsulation guarantees that the corresponding NETCONF messages are always well-formed XML documents.

For example, an instance of:

```
module example-config {
  yang-version 1.1;
  namespace "urn:example:config";
  prefix "co";

  container system { ... }
  container routing { ... }
}
```

could be encoded in NETCONF as:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <!-- system data here -->
      </system>
      <routing xmlns="urn:example:config">
        <!-- routing data here -->
      </routing>
    </config>
  </edit-config>
</rpc>
```

# Section 5.3.1 (YANG XML Namespace)

## 5.3.1. YANG XML Namespace

YANG defines an XML namespace for NETCONF `<edit-config>` operations, `<error-info>` content, and the `<action>` element. The name of this namespace is `"urn:ietf:params:xml:ns:yang:1"`.

# Section 5.6.4 (Announcing Conformance Information in NETCONF)

## 5.6.4. Announcing Conformance Information in NETCONF

This document defines the following mechanism for announcing conformance information. Other mechanisms may be defined by future specifications.

A NETCONF server MUST announce the modules it implements (see [Section 5.6.5](#)) by implementing the YANG module "ietf-yang-library" defined in [[RFC7895](#)] and listing all implemented modules in the "/modules-state/module" list.

The server also MUST advertise the following capability in the <hello> message (line breaks and whitespaces are used for formatting reasons only):

```
urn:ietf:params:netconf:capability:yang-library:1.0?  
  revision=<date>&module-set-id=<id>
```

The parameter "revision" has the same value as the revision date of the "ietf-yang-library" module implemented by the server. This parameter MUST be present.

The parameter "module-set-id" has the same value as the leaf "/modules-state/module-set-id" from "ietf-yang-library". This parameter MUST be present.

With this mechanism, a client can cache the supported modules for a server and only update the cache if the "module-set-id" value in the <hello> message changes.

# Section 7.5.8 (NETCONF <edit-config> Operations, for containers)

## 7.5.8. NETCONF <edit-config> Operations

Containers can be created, deleted, replaced, and modified through <edit-config> by using the "operation" attribute (see [Section 7.2 in \[RFC6241\]](#)) in the container's XML element.

If a container does not have a "presence" statement and the last child node is deleted, the NETCONF server MAY delete the container.

When a NETCONF server processes an <edit-config> request, the elements of procedure for the container node are as follows:

- o If the operation is "merge" or "replace", the node is created if it does not exist.
- o If the operation is "create", the node is created if it does not exist. If the node already exists, a "data-exists" error is returned.
- o If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

# Section 7.6.7 (NETCONF <edit-config> Operations, for leafs)

## 7.6.7. NETCONF <edit-config> Operations

When a NETCONF server processes an <edit-config> request, the elements of procedure for the leaf node are as follows:

- o If the operation is "merge" or "replace", the node is created if it does not exist, and its value is set to the value found in the XML RPC data.
- o If the operation is "create", the node is created if it does not exist. If the node already exists, a "data-exists" error is returned.
- o If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

# Section 7.7.9 (NETCONF <edit-config> Operations, for leaf-lists)

## 7.7.9. NETCONF <edit-config> Operations

Leaf-list entries can be created and deleted, but not modified, through <edit-config>, by using the "operation" attribute in the leaf-list entry's XML element.

In an "ordered-by user" leaf-list, the attributes "insert" and "value" in the YANG XML namespace ([Section 5.3.1](#)) can be used to control where in the leaf-list the entry is inserted. These can be used during "create" operations to insert a new leaf-list entry, or during "merge" or "replace" operations to insert a new leaf-list entry or move an existing one.

The "insert" attribute can take the values "first", "last", "before", and "after". If the value is "before" or "after", the "value" attribute MUST also be used to specify an existing entry in the leaf-list.

If no "insert" attribute is present in the "create" operation, it defaults to "last".

If several entries in an "ordered-by user" leaf-list are modified in the same <edit-config> request, the entries are modified one at a time, in the order of the XML elements in the request.

In a <copy-config> or in an <edit-config> with a "replace" operation that covers the entire leaf-list, the leaf-list order is the same as the order of the XML elements in the request.

<SNIP/>

# Section 7.7.10 (Usage Example) for leaf-lists

## 7.7.10. Usage Example

```
leaf-list allow-user {  
  type string;  
  description  
    "A list of user name patterns to allow."  
}
```

A corresponding XML instance example:

```
<allow-user>alice</allow-user>  
<allow-user>bob</allow-user>
```

To create a new element in this list, using the default <edit-config> operation "merge":

```
<rpc message-id="101"  
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <edit-config>  
    <target>  
      <running/>  
    </target>  
    <config>  
      <system xmlns="urn:example:config">  
        <services>  
          <ssh>  
            <allow-user>eric</allow-user>  
          </ssh>  
        </services>  
      </system>  
    </config>  
  </edit-config>  
</rpc>
```

<SNIP/>

# Section 7.8.6 (NETCONF <edit-config> Operations, for lists)

## 7.8.6. NETCONF <edit-config> Operations

List entries can be created, deleted, replaced, and modified through <edit-config> by using the "operation" attribute in the list's XML element. In each case, the values of all keys are used to uniquely identify a list entry. If all keys are not specified for a list entry, a "missing-element" error is returned.

In an "ordered-by user" list, the attributes "insert" and "key" in the YANG XML namespace ([Section 5.3.1](#)) can be used to control where in the list the entry is inserted. These can be used during "create" operations to insert a new list entry, or during "merge" or "replace" operations to insert a new list entry or move an existing one.

The "insert" attribute can take the values "first", "last", "before", and "after". If the value is "before" or "after", the "key" attribute MUST also be used, to specify an existing element in the list. The value of the "key" attribute is the key predicates of the full instance identifier (see [Section 9.13](#)) for the list entry.

<SNIP/>

# Section 7.8.7 (Usage Example) for lists

## 7.8.7. Usage Example

Given the following list:

```
list user {
  key "name";
  config true;
  description
    "This is a list of users in the system.";

  leaf name {
    type string;
  }
  leaf type {
    type string;
  }
  leaf full-name {
    type string;
  }
}
```

A corresponding XML instance example:

```
<user>
  <name>fred</name>
  <type>admin</type>
  <full-name>Fred Flintstone</full-name>
</user>
```

<SNIP/>

# Section 7.10.3 (NETCONF <edit-config> Operations, for anydata)

## 7.10.3. NETCONF <edit-config> Operations

An anydata node is treated as an opaque chunk of data. This data can be modified in its entirety only.

Any "operation" attributes present on subelements of an anydata node are ignored by the NETCONF server.

When a NETCONF server processes an <edit-config> request, the elements of procedure for the anydata node are as follows:

- o If the operation is "merge" or "replace", the node is created if it does not exist, and its value is set to the subelements of the anydata node found in the XML RPC data.
- o If the operation is "create", the node is created if it does not exist, and its value is set to the subelements of the anydata node found in the XML RPC data. If the node already exists, a "data-exists" error is returned.
- o If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

# Section 7.11.3 (NETCONF <edit-config> Operations, for anyxml)

## 7.11.3. NETCONF <edit-config> Operations

An anyxml node is treated as an opaque chunk of data. This data can be modified in its entirety only.

Any "operation" attributes present on subelements of an anyxml node are ignored by the NETCONF server.

When a NETCONF server processes an <edit-config> request, the elements of procedure for the anyxml node are as follows:

- o If the operation is "merge" or "replace", the node is created if it does not exist, and its value is set to the XML content of the anyxml node found in the XML RPC data.
- o If the operation is "create", the node is created if it does not exist, and its value is set to the XML content of the anyxml node found in the XML RPC data. If the node already exists, a "data-exists" error is returned.
- o If the operation is "delete", the node is deleted if it exists. If the node does not exist, a "data-missing" error is returned.

# Section 7.14.4 (NETCONF XML Encoding Rules, for rpc)

## 7.14.4. NETCONF XML Encoding Rules

An rpc node is encoded as a child XML element to the <rpc> element, as designated by the substitution group "rpcOperation" in [\[RFC6241\]](#). The element's local name is the rpc's identifier, and its namespace is the module's XML namespace (see [Section 7.1.3](#)).

Input parameters are encoded as child XML elements to the rpc node's XML element, in the same order as they are defined within the "input" statement.

If the RPC operation invocation succeeded and no output parameters are returned, the <rpc-reply> contains a single <ok/> element defined in [\[RFC6241\]](#). If output parameters are returned, they are encoded as child elements to the <rpc-reply> element defined in [\[RFC6241\]](#), in the same order as they are defined within the "output" statement.

# Section 7.14.5 (Usage Example, for rpc)

## 7.14.5. Usage Example

The following example defines an RPC operation:

```
module example-rock {
  yang-version 1.1;
  namespace "urn:example:rock";
  prefix "rock";

  rpc rock-the-house {
    input {
      leaf zip-code {
        type string;
      }
    }
  }
}
```

A corresponding XML instance example of the complete rpc and rpc-reply:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rock-the-house xmlns="urn:example:rock">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>

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

# Section 7.15.2 (NETCONF XML Encoding Rules, for action)

## 7.15.2. NETCONF XML Encoding Rules

When an action is invoked, an element with the local name "action" in the namespace "urn:ietf:params:xml:ns:yang:1" (see [Section 5.3.1](#)) is encoded as a child XML element to the <rpc> element defined in [\[RFC6241\]](#), as designated by the substitution group "rpcOperation" in [\[RFC6241\]](#).

The <action> element contains a hierarchy of nodes that identifies the node in the datastore. It MUST contain all containers and list nodes in the direct path from the top level down to the list or container containing the action. For lists, all key leafs MUST also be included. The innermost container or list contains an XML element that carries the name of the defined action. Within this element, the input parameters are encoded as child XML elements, in the same order as they are defined within the "input" statement.

Only one action can be invoked in one <rpc>. If more than one action is present in the <rpc>, the server MUST reply with a "bad-element" <error-tag> in the <rpc-error>.

If the action operation invocation succeeded and no output parameters are returned, the <rpc-reply> contains a single <ok/> element defined in [\[RFC6241\]](#). If output parameters are returned, they are encoded as child elements to the <rpc-reply> element defined in [\[RFC6241\]](#), in the same order as they are defined within the "output" statement.

# Section 7.15.3 (Usage Example, for action)

## 7.15.3. Usage Example

The following example defines an action to reset one server at a server farm:

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

  import ietf-yang-types {
    prefix "yang";
  }

  list server {
    key name;
    leaf name {
      type string;
    }
    action reset {
      input {
        leaf reset-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

<SNIP/>

# Section 7.16.2 (NETCONF XML Encoding Rules, for notification)

## 7.16.2. NETCONF XML Encoding Rules

A notification node that is defined on the top level of a module is encoded as a child XML element to the <notification> element defined in "NETCONF Event Notifications" [[RFC5277](#)]. The element's local name is the notification's identifier, and its namespace is the module's XML namespace (see [Section 7.1.3](#)).

Bjorklund

Standards Track

[Page 117]

---

RFC 7950

YANG 1.1

August 2016

When a notification node is defined as a child to a data node, the <notification> element defined in [[RFC5277](#)] contains a hierarchy of nodes that identifies the node in the datastore. It MUST contain all containers and list nodes from the top level down to the list or container containing the notification. For lists, all key leafs MUST also be included. The innermost container or list contains an XML element that carries the name of the defined notification.

The notification's child nodes are encoded as subelements to the notification node's XML element, in any order.

# Section 7.16.3 (Usage Example, for notification)

## 7.16.3. Usage Example

The following example defines a notification at the top level of a module:

```
module example-event {
  yang-version 1.1;
  namespace "urn:example:event";
  prefix "ev";

  notification event {
    leaf event-class {
      type string;
    }
    leaf reporting-entity {
      type instance-identifier;
    }
    leaf severity {
      type string;
    }
  }
}
```

A corresponding XML instance example of the complete notification:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-07-08T00:01:00Z</eventTime>
  <event xmlns="urn:example:event">
    <event-class>fault</event-class>
    <reporting-entity>
      /ex:interface[ex:name='Ethernet0']
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

<SNIP/>

Questions, Concerns, Comments?