Introduction to NETCONF and YANG

Jürgen Schönwälder

http://www.eecs.jacobs-university.de/users/schoenw/

June 5, 2008
## Overview

### Structure of the Tutorial

<table>
<thead>
<tr>
<th>Part</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1: Motivation and Background</td>
<td>20 min</td>
</tr>
<tr>
<td>Part 2: NETCONF Protocol</td>
<td>30 min</td>
</tr>
<tr>
<td>Part 3: Practice NETCONF</td>
<td>50 min</td>
</tr>
<tr>
<td>Part 4: YANG Data Modeling Language</td>
<td>30 min</td>
</tr>
<tr>
<td>Part 5: Practice YANG</td>
<td>50 min</td>
</tr>
</tbody>
</table>
Part: Motivation and Background

1. Configuration Management Approaches

2. Configuration Management Requirements

3. Internet Management Framework
Part: Motivation and Background

1. Configuration Management Approaches

2. Configuration Management Requirements

3. Internet Management Framework
"The Network is the Record" Approach

- Labor intensive, expensive, error prone, widely deployed
All configuration changes are made (and validated) on the network-wide configuration database and devices are never touched manually.
Part: Motivation and Background

1. Configuration Management Approaches

2. Configuration Management Requirements

3. Internet Management Framework
R1: configuration state vs. operational state
A configuration management protocol must be able to distinguish between configuration state and operational state.

R2: concurrency support
A configuration management protocol must provide primitives to prevent errors due to concurrent configuration changes.

R3: configuration transactions
A configuration management protocol must provide primitives to apply configuration changes to a set of network elements in a robust and transaction-oriented way.
R4: distribution vs. activation

It is important to distinguish between the distribution of configurations and the activation of a certain configuration.

R5: distinguish multiple configurations

A configuration management protocol must be able to distinguish between several configurations and devices should be able to hold multiple configurations.

R6: persistence of configuration state

A configuration management protocol must be clear about the persistence of configuration changes.
### R7: configuration change events

A configuration management protocol must be able to report configuration change events to help tracing back configuration changes.

### R8: configuration dump and restore

A full configuration dump and a full configuration restore are primitive operations frequently used by operators and must be supported appropriately.
R9: support for standard tools

A configuration management protocol must represent configuration state and operational state in a form which allows operators to use existing comparison, conversion, and versioning tools.

R10: minimize impact of configuration changes

Configurations must be described such that devices can determine a set of operations to bring the devices from a given configuration state to the desired configuration state, minimizing the impact caused by the configuration change itself on networks and systems.
Part: Motivation and Background

1. Configuration Management Approaches

2. Configuration Management Requirements

3. Internet Management Framework
RFC 3410

The purpose of this document is to provide an overview of the third version of the Internet-Standard Management Framework, termed the SNMP version 3 Framework (SNMPv3).

Consequences

- WG's were forced to produce SNMP MIB modules
- Small group of people contributing to NM technologies
- Almost no operator involvement (they used CLIs, SYSLOG, SNMP, . . .)
Today’s IETF view

\[ \text{NM} = \text{SNMP} + \text{SYSLOG} + \text{NETCONF} + \text{IPFIX} + \ldots \]

Took years to change this view...

- COPS vs. SNMP debate (2000)
- IETF NM road show (2000-2001)
- IAB NM workshop in 2002
- Lots of (repeated) discussions
J. Schönewälder.

L. Sanchez, K. McCloghrie, and J. Saperia.
Requirements for Configuration Management of IP-based Networks.

J. Schönewälder.
Overview of the 2002 IAB Network Management Workshop.

Understanding BGP Misconfiguration.

D. Oppenheimer, A. Ganapathi, and D. A. Patterson.
Why do Internet services fail, and what can be done about it?
4 Architectural Aspects

5 Capability Exchange and Remote Procedure Calls

6 Protocol Operations

7 Transport Mappings
NETCONF IETF Working Group

Milestones

- NETCONF WG chartered in May 2003, core specifications published in December 2006
- Heavily influenced by Juniper’s JunoScript
- Core contributors from Juniper Networks and Cisco
- Some design decisions were difficult to take

Status

The NETCONF WG is still active:

- notifications
- fine grained locking
- NETCONF monitoring
- TLS transport
Part: NETCONF

4 Architectural Aspects

5 Capability Exchange and Remote Procedure Calls

6 Protocol Operations

7 Transport Mappings
- NETCONF enabled devices include a NETCONF server
- Management applications include a NETCONF client
- Command Line Interfaces (CLIs) can be a wrapped around a NETCONF client
Security has to be provided by the transport layer.
The operations layer provides the primitives to handle configurations.
The set of operations is supposed to be extensible.
- Does not exist formally (so take this with some care)
- Loosely based on SNMP architectural concepts
A configuration datastore is the complete set of configuration information that is required to get a device from its initial default state into a desired operational state.

- The <running> configuration datastore represents the currently active configuration of a device and is always present.
- The <startup> configuration datastore represents the configuration that will be used during the next startup.
- The <candidate> configuration datastore represents a configuration that may become a <running> configuration through an explicit commit.
Transaction Models

- **Direct Model**
  - `<edit-config>` → `running`

- **Candidate Model (optional)**
  - `<edit-config>` → `candidate` → `<commit>` → `running`

- **Distinct Startup Model (optional)**
  - `<edit-config>` → `running` → `<commit>` → `<copy-config>` → `startup`

- Some operations (`edit-config`) may support different error behaviours, including rollback behaviour.
Part: NETCONF

4 Architectural Aspects

5 Capability Exchange and Remote Procedure Calls

6 Protocol Operations

7 Transport Mappings
Hello

After establishing a (secure) transport, both NETCONF protocol engines send a hello message to announce their capabilities and the session identifier.

A: <hello>
A:  <capabilities>
A:   <capability>
A:   </capability>
A:   <capability>
A:     urn:ietf:params:xml:ns:netconf:base:1.0#startup
A:   </capability>
A:  </capabilities>
A:  <session-id>4</session-id>
A: </hello>
The Remote Procedure Call (RPC) protocol consists of a <rpc/> message followed by an <rpc-reply/> message.

M: <rpc message-id="101"
M:     <get-config>
M:       <source>
M:         <running/>
M:       </source>
M:     </get-config>
M: </rpc>
A: <rpc-reply message-id="101"
A:     <data><!-- ...contents here... --></data>
A: </rpc-reply>
RPC failures are indicated by one or more `<rpc-error/>` elements in the `<rpc-reply/>` element.

M:   <get-config><source><running/></source></get-config>
M: </rpc>
A:   <rpc-error>
A:     <error-type>rpc</error-type>
A:     <error-tag>missing-attribute</error-tag>
A:     <error-severity>error</error-severity>
A:     <error-info>
A:       <bad-attribute>message-id</bad-attribute>
A:       <bad-element>rpc</bad-element>
A:     </error-info>
A:   </rpc-error>
A: </rpc-reply>`
Part: NETCONF

4 Architectural Aspects

5 Capability Exchange and Remote Procedure Calls

6 Protocol Operations

7 Transport Mappings
NETCONF Operations Overview

The diagram illustrates the operations and responses in the NETCONF protocol, showing how commands are exchanged between a Core Cmd Generator and a Core Cmd Responder. Each operation is represented by a command, followed by a response or notification, indicating the flow of communication in a session.
NETCONF Operations

- get-config(source, filter)
  Retrieve a (filtered subset of a) configuration from the configuration datastore source.

- edit-config(target, default-operation, test-option, error-option, config)
  Edit the target configuration datastore by merging, replacing, creating, or deleting new config elements.

- copy-config(target, source)
  Copy the content of the configuration datastore source to the configuration datastore target.

- delete-config(target)
  Delete the named configuration datastore target.
lock(target)
Lock the configuration datastore target.

unlock(target)
Unlock the configuration datastore target.

get(filter)
Retrieve (a filtered subset of a) the running configuration and device state information.

close-session()
Gracefully close the current session.

kill-session(session)
Force the termination of the session session.
commit()
Commit candidate configuration datastore to the running configuration (#candidate capability).

discard-changes()
Revert the candidate configuration datastore to the running configuration (#candidate capability).

validate(source)
Validate the contents of the configuration datastore source (#validate capability).

create-subscription(stream, filter, start, stop)
Subscribe to a notification stream with a given filter and the start and stop times.
**Editing Configuration**

- **merge**
  The configuration data is merged with the configuration at the corresponding level in the configuration datastore.

- **replace**
  The configuration data replaces any related configuration in the configuration datastore identified by the target parameter.

- **create**
  The configuration data is added to the configuration if and only if the configuration data does not already exist.

- **delete**
  The configuration data identified by the element containing this attribute is deleted in the configuration datastore.
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
       <target>
         <running/>
       </target>
       <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
         <top xmlns="http://example.com/schema/1.2/config">
           <interface xc:operation="replace">
             <name>Ethernet0/0</name>
             <mtu>1500</mtu>
             <address>
               <name>192.0.2.4</name>
               <prefix-length>24</prefix-length>
             </address>
           </interface>
         </top>
       </config>
     </edit-config>
   </rpc>
Subtree Filtering

**Subtree Filter Expressions**

Subtree filter expressions select particular XML subtrees to include in `get` and `get-config` responses.

**Namespace Selection**

If the 'xmlns' attribute is present, then the filter output will only include elements from the specified namespace.

**Attribute Match Expressions**

The set of (unqualified or qualified) XML attributes present in any type of filter node form an “attribute match expression” The selected data must have matching values for every attribute of an attribute match expression.
**Containment Nodes**

For each containment node specified in a subtree filter, all data model instances must exactly match the specified namespaces, element hierarchy, and any attribute match expressions.

**Selection Nodes**

An empty leaf node within a filter is called a “selection node” and it selects the specified subtree(s) and it suppresses the automatic selection of the entire set of sibling nodes in the underlying data model.

**Content Match Nodes**

A leaf node that contains simple content is called a “content match node” and it selects some or all of its sibling nodes. It represents an exact-match filter on the leaf node element content.
Subtree Filtering Example

<filter type="subtree">

<!-- namespace selection and containment node selection -->
<t:top xmlns:t="http://example.com/schema/1.2/config">

<!-- containment node selection -->
<t:interfaces>

<!-- containment node selection and attribute match expression -->
<t:interface t:ifName="eth0">

<!-- selection node -->
<t:ifSpeed/>

<!-- content match node -->
<t:ifType>Ethernet</t:if-type>

</t:interface>
</t:interfaces>
</t:top>
</filter>
Part: NETCONF

4 Architectural Aspects

5 Capability Exchange and Remote Procedure Calls

6 Protocol Operations

7 Transport Mappings
SSH Protocol

- SSH is a protocol for secure remote login and other secure network services over an insecure network.
- The SSH protocol consists of three major components:
  1. The *Transport Layer Protocol* provides server authentication, confidentiality, and integrity with perfect forward secrecy.
  2. The *User Authentication Protocol* authenticates the client-side user to the server.
  3. The *Connection Protocol* multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.
- SSH is widely deployed on network devices as a secure protocol to access the command line interface.
Motivation: Use an already deployed security protocol, thereby reducing the operational costs associated with key management.

SSH supports multiple logical channels over one transport layer association.

For framing purposes, the special end of message marker `]]]>` has been introduced.

NETCONF over SSH has been selected as the mandatory to implement transport for NETCONF.
A: <?xml version="1.0" encoding="UTF-8"?>
A: <hello>
A:  <capabilities>
A:   <capability>
A:   </capability>
A:   <capability>
A:    urn:ietf:params:xml:ns:netconf:base:1.0#startup
A:   </capability>
A: </capabilities>
A: <session-id>4</session-id>
A: </hello>
A: ]]>]]>
M: <xml version="1.0" encoding="UTF-8"/>
M: <hello>
M:   <capabilities>
M:     <capability>
M:     </capability>
M:   </capabilities>
M: </hello>
M: ]]>]]>
<xml version="1.0" encoding="UTF-8"?>
<rpc message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <config xmlns="http://example.com/schema/1.2/config">
        <users/>
      </config>
    </filter>
  </get-config>
</rpc>
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <config xmlns="http://example.com/schema/1.2/config">
      <users>
        <user><name>root</name><type>superuser</type></user>
        <user><name>fred</name><type>admin</type></user>
        <user><name>barney</name><type>admin</type></user>
      </users>
    </config>
  </data>
</rpc-reply>
BEEP Protocol (RFC 3080)

- BEEP is a generic application protocol kernel for connection-oriented, asynchronous interactions.
- BEEP supports multiple channels, application layer framing and fragmentation.
- BEEP exchange styles:
  - MSG/RPY
  - MSG/ERR
  - MSG/ANS
- Integrates into SASL (RFC 2222) and TLS (RFC 2246) for security.
- Connections can be initiated by both participating peers (no strict client/server roles).
BEEP supports multiple logical channels.
Every peer can be the initiator of a connection.
SASL allows to map to existing security infrastructures.
Framing and fragmentation services provided by BEEP.
BEEP is currently not widely deployed and there is a lack of operational experience with BEEP in the operator community.
BEEP is an optional NETCONF transport.
M: MSG 0 1 . 10 48 101
M: Content-Type: application/beep+xml
M: <start number='''1'''>
M:  <profile uri='''http://iana.org/beep/netconf''' />
M: </start>
M: END
A: RPY 0 1 . 38 87
A: Content-Type: application/beep+xml
A:
A: <profile uri='''http://iana.org/beep/netconf''' />
A: END
A: MSG 1 0 . 0 436
A: Content-Type: application/beep+xml
A:
A: <capabilities>
A: <capability>
A: </capability>
A: <capability>
A: urn:ietf:params:xml:ns:netconf:base:1.0#startup
A: </capability>
A: </capabilities>
A: <session-id>4</session-id>
A: </hello>
A: END
M: RPY 1 0 . 0 0
M: END
M: MSG 1 42 . 24 344
M: Content-Type: text/xml; charset=utf-8
M:
M: <rpc message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
M:  <get-config>
M:   <source>
M:    <running/>
M:   </source>
M:  </get-config>
M:  <config xmlns="http://example.com/schema/1.2/config">
M:    <users/>
M:  </config>
M: </filter>
M: </get-config>
M: </rpc>
M: END
NETCONF over BEEP Example

A: RPY 1 42 . 24 542
A: Content-Type: text/xml; charset=utf-8
A:
A:   <data>
A:     <config xmlns="http://example.com/schema/1.2/config">
A:       <users>
A:         <user><name>root</name><type>superuser</type></user>
A:         <user><name>fred</name><type>admin</type></user>
A:         <user><name>barney</name><type>admin</type></user>
A:       </users>
A:     </config>
A:   </data>
A: </rpc-reply>
A: END
Instead of inventing a special purpose RPC protocol, use existing Web Services standards.

**Pros:**
- more developers / tools available
- better integration with IT infrastructure

**Cons:**
- base technology not under control of the IETF
- unneeded complexity
- interoperability problems (immature technology)
- HTTP is a bad generic application protocol kernel

**Note:** Transport mapping does not map NETCONF operations to SOAP operations!
M: POST /netconf HTTP/1.1
M: Host: netconfdevice
M: Content-Type: text/xml; charset=utf-8
M: Accept: application/soap+xml, text/*
M: Cache-Control: no-cache
M: Pragma: no-cache
M: Content-Length: 490
M:
M: <?xml version="1.0" encoding="UTF-8"?>
M: <soapenv:Envelope
M: xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
M: <soapenv:Body>
M: <rpc message-id="101"
M: <get-config
M: <source><running/></source>
M: <filter type="subtree">
M: <top xmlns="http://example.com/schema/1.2/config">
M: <users/>
M: </top>
M: </filter>
M: </get-config
M: </rpc>
M: </soapenv:Body>
M: </soapenv:Envelope>
A: HTTP/1.1 200 OK
A: Content-Type: application/soap+xml; charset=utf-8
A: Content-Length: 668
A:
A: <?xml version="1.0" encoding="UTF-8"?>
A: <soapenv:Envelope
A: xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
A: <soapenv:Body>
A: <rpc-reply message-id="101"
A: <data>
A: <top xmlns="http://example.com/schema/1.2/config">
A: <users>
A: <user>
A: <name>root</name>
A: <type>superuser</type>
A: <full-name>Charlie Root</full-name>
A: <dept>1</dept>
A: <id>1</id>
A: <company-info>
A: </user>
A: </users>
A: </top>
A: </data>
A: </rpc-reply>
A: </soapenv:Body>
A: </soapenv:Envelope>
R. Enns.
NETCONF Configuration Protocol.
RFC 4741, Juniper Networks, December 2006.

M. Wasserman and T. Goddard.
Using the NETCONF Configuration Protocol over Secure SHell (SSH).
RFC 4742, ThingMagic, ICEsoft Technologies, December 2006.

T. Goddard.
Using NETCONF over the Simple Object Access Protocol (SOAP).
RFC 4743, ICEsoft Technologies, December 2006.

E. Lear.
Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP).
RFC 4744, Cisco Systems, December 2006.

S. Chisholm and H. Trevino.
NETCONF Event Notifications.
Overview

Modules

Built-in types and derived types

Leafs, Leaf-lists, Container, Lists

Tools
YANG’s purpose

YANG is an extensible NETCONF data modeling language able to model configuration data, state data, operations, and notifications. YANG definitions directly map to XML content.

YANG vs. YIN

YANG uses a compact SMIng like syntax since readability is highest priority. YIN is an XML version of YANG (lossless roundtrip conversion).

YANG vs. XSD or RELAX NG

YANG can be translated to XML Schema (XSD) and RELAX NG so that existing tools can be utilized.
### YANG Milestones (pre IETF)
- YANG design team created in Spring 2007
- Three design team meetings (USA, London, Stockholm)
- YANG discussions at the 71st IETF (Vancouver)
- YANG discussions at the 72nd IETF (Philadelphia)

### NETMOD Milestones
- Apr. 2008: NETMOD WG chartered
- Aug. 2008: initial YANG, YIN, DSDL, ... documents
- Mar. 2009: submit architecture document to the IESG
- Sep. 2009: submit YANG, YIN, DSDL, ... to the IESG
Overview

Modules

Built-in types and derived types

Leafs, Leaf-lists, Container, Lists

Tools
Modules and submodules

Module
A self-contained collection of YANG definitions.

Submodule
A partial module definition which contributes derived types, groupings, data nodes, RPCs, and notifications to a module.
module acme-module {
    namespace "http://acme.example.com/module";
    prefix "acme";

    import "yang-types" {
        prefix "yang";
    }
    include "acme-system";

    organization "ACME Inc.";
    contact    "support@acme.example.com";
    description
        "The module for entities implementing the ACME products";

    revision "2007-06-09" {
        description "Initial revision.";
    }
}

8 Overview

9 Modules

10 Built-in types and derived types

11 Leafs, Leaf-lists, Container, Lists

12 Tools
## Built-in Data Types

<table>
<thead>
<tr>
<th>Category</th>
<th>Types</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral</td>
<td>{u,}int{8,16,32,64}</td>
<td>range</td>
</tr>
<tr>
<td>Floats</td>
<td>float{32,64}</td>
<td>range</td>
</tr>
<tr>
<td>String</td>
<td>string</td>
<td>length, pattern</td>
</tr>
<tr>
<td>Enumeration</td>
<td>enumeration</td>
<td>enum</td>
</tr>
<tr>
<td>Bool and Bits</td>
<td>boolean, bits</td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td>binary</td>
<td>length</td>
</tr>
<tr>
<td>References</td>
<td>keyref</td>
<td></td>
</tr>
<tr>
<td>References</td>
<td>instance-identifier</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>empty</td>
<td></td>
</tr>
</tbody>
</table>

### Type system

The data type system is mostly an extension of the SMIng type system, accommodating XML and XSD requirements.
module inet-types {

    namespace "urn:ietf:params:xml:ns:yang:inet-types";
    prefix "inet";

    typedef ipv4-address {
        type string {
            pattern '(([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}'
            + '([0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])'
            + '(%[\p{N}\p{L}]*)?';
        }
    }

    typedef ip-address {
        type union {
            type inet:ipv4-address;
            type inet:ipv6-address;
        }
        description "Represents a version neutral IP address.";
    }
}
Part: YANG

8 Overview

9 Modules

10 Built-in types and derived types

11 Leafs, Leaf-lists, Container, Lists

12 Tools
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaf</td>
<td>A leaf has one value, no children, one instance.</td>
</tr>
<tr>
<td>leaf-list</td>
<td>A leaf-list has one value, no children, multiple instances.</td>
</tr>
<tr>
<td>container</td>
<td>A container has no value, holds related children, has one instance.</td>
</tr>
<tr>
<td>list</td>
<td>A list has no value, holds related children, has multiple instances, has a key property.</td>
</tr>
</tbody>
</table>
Example: leaf and leaf-list

leaf domain {
    type inet:domain-name; // values are typed (type imported)
    mandatory true;       // must exist in a valid configuration
    config true;         // part of the set of configuration objects
    description
        "The host name of this system.";
}

// XML: <domain>example.com</domain>

leaf-list search {
    type inet:domain-name; // imported from the module with prefix inet
    ordered-by user;      // maintain the order given by the user
    description
        "List of domain names to search.";
}

// XML: <search>eng.example.com</search>
// XML: <search>example.com</search>
container system {
    config true;
    leaf hostname {
        type inet:domain-name;
    }
}

container resolver {
    leaf domain { /* see above */ }
    leaf-list search { /* see above */ }
    description
        "The configuration of the resolver library."
}

// XML: <system>
// XML: <hostname>server.example.com</hostname>
// XML: <resolver>
// XML:  <domain>example.com</domain>
// XML:  <search>eng.example.com</search>
// XML:  <search>example.com</search>
// XML: </resolver>
// XML: </system>
list nameserver {
    key address;
    leaf address {
        type inet:ip-address;
    }
    leaf status {
        type enumeration {
            enum enabled; enum disabled; enum failed;
        }
    }
}

// XML:   <nameserver>
// XML:       <address>192.0.2.1</address>
// XML:       <status>enabled</status>
// XML:   </nameserver>
// XML:   <nameserver>
// XML:       <address>192.0.2.2</address>
// XML:       <status>failed</status>
// XML:   </nameserver>
**Augment, Must, When**

**augment**

The *augment* statement can be used to place nodes into an existing hierarchy using the current module’s namespace.

**must**

The *must* statement can be used to express constraints (in the form of XPATH expressions) that must be satisfied by a valid configuration.

**when**

The *when* statement can be used to define sparse augmentations where nodes are only added when a condition (expressed in the form of an XPATH expression) is true.
augment system/resolver {
    container debug {
        presence "enables debugging";
        description
            "This container enables debugging.";
        leaf level {
            type enumeration {
                enum low;
                enum medium;
                enum full;
            }
            default "medium";
            mandatory false;
            description
                "The debugging level; default is medium debug information.";
        }
    }
}

// XML: <system><resolver>
// XML:   <debug/>
// XML: </resolver></system>
augment system/resolver {
    leaf access-timeout {
        type uint32;
        unit "seconds";
        mandatory true;
        description "Maximum time without server response.";
    }
    leaf retry-timer {
        type uint32;
        units "seconds";
        description "Period after which to retry an operation";
        must "this < ../access-timeout" {
            error-app-tag "retry-timer-invalid";
            error-message "The retry timer must be less "
            + "than the access timeout";
        }
    }
}
Example: augment and when

```yml
augment system/resolver/nameserver {
  when "status = enabled";
  leaf tx {
    type yang:counter32;
    config false;
  }
  leaf rx {
    type yang:counter32;
    config false;
  }
}
```

// XML:      <nameserver>
// XML:    <address>192.0.2.1</address>
// XML:    <status>enabled</status>
// XML:    <tx>2345</tx>
// XML:    <rx>1234</rx>
// XML:  </nameserver>

// XML:      <nameserver>
// XML:    <address>192.0.2.2</address>
// XML:    <status>failed</status>
// XML:  </nameserver>
```
Grouping and Choice

**grouping**

A grouping is a reusable collection of nodes. The grouping mechanism can be used to emulate structured data types or objects. A grouping can be refined when it is used.

**choice**

A choice allows one alternative of the choice to exist. The choice mechanism can be used to provide extensibility hooks that can be exploited using augments.

- Should a grouping be considered a template mechanism or a structured data type mechanism?
Example: grouping

grouping target {
  leaf address {
    type inet:ip-address;
    description "Target IP address.";
  }
  leaf port {
    type inet:ip-port;
    description "Target port number.";
  }
}

list nameserver {
  key "address port";
  uses target;
}

// XML: <nameserver>
// XML:   <address>192.0.2.1</address>
// XML:   <port>53</port>
// XML: </nameserver>
container transfer {
    choice how {
        default interval;
        case interval {
            leaf interval {
                type uint16; default 30; units minutes;
            }
        }
        case daily {
            leaf daily {
                type empty;
            }
            leaf time-of-day {
                type string; units 24-hour-clock; default 1am;
            }
        }
        case manual {
            leaf manual {
                type empty;
            }
        }
    }
}
The `notification` statement can be used to define the contents of notifications.

The `rpc` statement can be used to define operations together with their input and output parameters carried over the RPC protocol.

- Should the `rpc` statement be called `operation` since it is used to define operations?
- Should all NETCONF operations be formally defined in YANG?
notification nameserver-failure {
  description
      "A failure of a nameserver has been detected and the server has been disabled."
  leaf address {
    type keyref {
      path "/system/resolver/nameserver/address";
    }
  }
}

// PROT: <notification>
// PROT: <eventTime>2008-06-03T18:34:50+02:00</eventTime>
// PROT: <nameserver-failure>
// PROT: <address>192.0.2.2</address>
// PROT: </nameserver-failure>
// PROT: </notification>
rpc activate-software-image {
  input {
    leaf image name {
      type string;
    }
  }
  output {
    leaf status {
      type string;
    }
  }
}

// RPC: <rpc xmlns="urn:mumble" message-id="42">
// RPC: <activate-software-image>
// RPC: <image-name>image.tgz</image-name>
// RPC: </activate-software-image>
// RPC: </rpc>
Overview

Modules

Built-in types and derived types

Leafs, Leaf-lists, Container, Lists

Tools
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyang</td>
<td>Open source YANG validator and translator written in Python.</td>
</tr>
<tr>
<td>yangdump</td>
<td>Closed source YANG validator and translator written in C.</td>
</tr>
<tr>
<td>smidump</td>
<td>Open source SMI to YANG translator written in C.</td>
</tr>
<tr>
<td>emacs</td>
<td>Open source YANG editing mode for the emacs editor.</td>
</tr>
</tbody>
</table>
M. Bjorklund.
YANG - A data modeling language for NETCONF.

J. Schönwälder.
Common YANG Data Types.
Internet Draft (work in progress) <draft-schoenw-netmod-yang-types-00.txt>, Jacobs University, May 2008.

J. Schönwälder.
Protocol Independent Network Management Data Modeling Languages - Lessons Learned from the SMIng Project.