

# NETCONF and YANG Status, Tutorial, Demo

Jürgen Schönwälder



JACOBS  
UNIVERSITY

75th IETF 2009, Stockholm, 2009-07-30

This tutorial was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

# What is NETCONF?

NETCONF is a network management protocol specifically designed to support configuration management. It provides the following features:

- distinction between configuration and state data
- multiple configuration datastores (running, startup, ...)
- support for configuration change transactions
- configuration testing and validation support
- selective data retrieval with filtering
- streaming and playback of event notifications
- extensible remote procedure call mechanism

# Cool — and what is YANG?

YANG is a data modeling language designed to write (configuration) data models for the NETCONF protocol. It provides the following features:

- human readable easy to learn representation
- hierarchical configuration data models
- reusable types and groupings
- extensibility through augmentation mechanisms
- supports the definition of operations (RPCs)
- formal constraints for configuration validation
- data model modularity through features / sub-modules
- versioning rules and development support
- translations to XSD, RelaxNG and YIN

# NETCONF and YANG Timeline

## Timeline

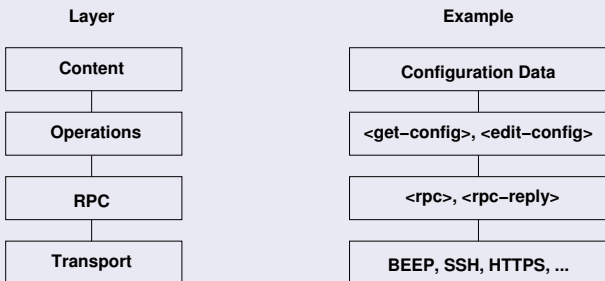
Jun 2002	IAB network management workshop
May 2003	NETCONF WG established
Dec 2006	NETCONF core RFCs published
2007	YANG design team creates YANG proposal
Apr 2008	NETMOD WG established
2009	YANG core RFCs published

## IETF WGs considering YANG

- IP Flow Information Export (IPFIX)
- ...

# NETCONF Layering Model (RFC4741)

Layers. . .



The one and only most popular NETCONF figure. . .

**STOP**

```
$ ssh -s broccoli netconf
<?xml version="1.0" encoding="UTF-8"?>
<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:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:url:1.0?scheme=ftp,file</capabil
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capabilit
<capability>http://tail-f.com/ns/aaa/1.1</capability>
<capability>http://tail-f.com/ns/execd/1.1</capability>
<capability>urn:ietf:params:xml:ns:yang:inet-types?revision=2009-05-13</capabil
<capability>urn:ietf:params:xml:ns:yang:yang-types?revision=2009-05-13</capabil
</capabilities>
<session-id>123</session-id></hello>]]>]]>

<?xml version="1.0" encoding="UTF-8"?>
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
  </capabilities>
</hello>]]>]]>
```

```
#!/usr/bin/env python2.6
#
# Connect to the NETCONF server passed on the command line and
# display their capabilities. This script and the following scripts
# all assume that the user calling the script is known by the server
# and that suitable SSH keys are in place. For brevity and clarity
# of the examples, we omit proper exception handling.
#
# $ ./nc01.py broccoli

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user):
    with manager.connect(host=host, port=22, username=user) as m:
        for c in m.server_capabilities:
            print c

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"))
```



```
#!/usr/bin/env python2.6
#
# Retrieve the running config from the NETCONF server passed on the
# command line using get-config and write the XML configs to files.
#
# $ ./nc02.py broccoli

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user):
    with manager.connect(host=host, port=22, username=user) as m:
        c = m.get_config(source='running').data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"))
```

```
#!/usr/bin/env python2.6
#
# Retrieve a portion selected by an XPATH expression from the running
# config from the NETCONF server passed on the command line using
# get-config and write the XML configs to files.
#
# $ ./nc03.py broccoli "aaa/authentication/users/user[name='schoenw']"

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, expr):
    with manager.connect(host=host, port=22, username=user) as m:
        assert("xpath" in m.server_capabilities)
        c = m.get_config(source='running', filter=('xpath', expr)).data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2])
```

# CONTINUE

# 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.

# NETCONF Operations (cont.)

- `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`.

# NETCONF Operations (cont.)

- `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).
- `commit(confirmed, confirm-timeout)`  
Commit candidate configuration datastore to the running configuration (:candidate 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.

**STOP**



```
#!/usr/bin/env python2.6
#
# Create a new user to the running configuration using edit-config
# and the test-option provided by the :validate capability.
#
# $ ./nc04.py broccoli bob 42 42

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, name, uid, gid):
    snippet = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
        <authentication> <users> <user xc:operation="create">
          <name>%s</name> <uid>%s</uid> <gid>%s</gid>
          <password>*</password> <ssh_keydir/> <homedir/>
        </user></users></authentication></aaa></config>""" % (name, uid, gid)

    with manager.connect(host=host, port=22, username=user) as m:
        assert(":validate" in m.server_capabilities)
        m.edit_config(target='running', config=snippet,
                      test_option='test-then-set')

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2], sys.argv[3], sys.argv[4])
```

```
#!/usr/bin/env python2.6
#
# Delete an existing user from the running configuration using
# edit-config and the test-option provided by the :validate
# capability.
#
# $ ./nc05.py broccoli bob

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, name):
    snippet = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
        <authentication> <users> <user xc:operation="delete">
            <name>%s</name>
        </user></users></authentication></aaa></config>""" % name

    with manager.connect(host=host, port=22, username=user) as m:
        assert(":validate" in m.server_capabilities)
        m.edit_config(target='running', config=snippet,
                      test_option='test-then-set')

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2])
```

# CONTINUE

# Configuration Datastores

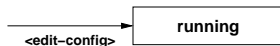
## Definition

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.

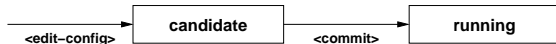
- The `<running>` configuration datastore represents the currently active configuration of a device and is always present.
- The `<startup>` configuration datastore represents the configuration that will be used during the next startup.
- The `<candidate>` configuration datastore represents a configuration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

# Transaction Models

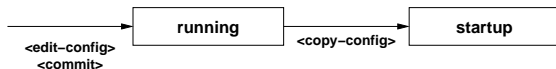
## Direct Model



## Candidate Model (optional)



## Distinct Startup Model (optional)



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

**STOP**

```
#!/usr/bin/env python2.6
#
# Delete a list of existing users from the running configuration using
# edit-config; protect the transaction using a lock.
#
# $ ./nc06.py broccoli bob alice

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

template = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
    <authentication> <users> <user xc:operation="delete">
      <name>%s</name> </user></users></authentication></aaa></config>"""

def demo(host, user, names):
    with manager.connect(host=host, port=22, username=user) as m:
        with m.locked(target='running'):
            for n in names:
                m.edit_config(target='running', config=template % n)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2:])
```

```
#!/usr/bin/env python2.6
#
# Delete a list of existing users from the running configuration using
# edit-config and the candidate datastore protected by a lock.
#
# $ ./nc07.py broccoli bob alice

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

template = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
    <authentication> <users> <user xc:operation="delete">
      <name>%s</name> </user></users></authentication></aaa></config>"""

def demo(host, user, names):
    with manager.connect(host=host, port=22, username=user) as m:
        assert(":candidate" in m.server_capabilities)
        m.discard_changes()
        with m.locked(target='candidate'):
            for n in names:
                m.edit_config(target='candidate', config=template % n)
            m.commit()

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2:])
```



# CONTINUE

# NETCONF Implementations

## Commercial Toolkits (not necessarily complete)

Applied Informatics	<a href="http://www.appinf.com/">&lt;http://www.appinf.com/&gt;</a>
Netconf Central	<a href="http://www.netconfcentral.org/">&lt;http://www.netconfcentral.org/&gt;</a>
SNMP Research	<a href="http://www.snmp.com/">&lt;http://www.snmp.com/&gt;</a>
Silicon and Software Systems	<a href="http://embeddedmind.com/">&lt;http://embeddedmind.com/&gt;</a>
Tail-f (ConfD)	<a href="http://www.tail-f.com/">&lt;http://www.tail-f.com/&gt;</a>

## Device Vendors (not necessarily complete)

Alaxala	<a href="http://www.alaxala.com/">&lt;http://www.alaxala.com/&gt;</a>
Cisco Systems	<a href="http://www.cisco.com/">&lt;http://www.cisco.com/&gt;</a>
Ericsson	<a href="http://www.ericsson.com/">&lt;http://www.ericsson.com/&gt;</a>
Juniper Networks	<a href="http://www.juniper.net/">&lt;http://www.juniper.net/&gt;</a>
Nortel	<a href="http://www.nortel.com/">&lt;http://www.nortel.com/&gt;</a>
RuggedCom	<a href="http://www.ruggedcom.com/">&lt;http://www.ruggedcom.com/&gt;</a>
Taseon	<a href="http://www.taseon.com/">&lt;http://www.taseon.com/&gt;</a>
Verivue	<a href="http://www.verivue.com/">&lt;http://www.verivue.com/&gt;</a>

# NETCONF Implementations (cont.)

## Open Source (not necessarily complete)

YencaP	<a href="http://ensuite.sourceforge.net/">&lt;http://ensuite.sourceforge.net/&gt;</a>
ncclient	<a href="http://code.google.com/p/ncclient/">&lt;http://code.google.com/p/ncclient/&gt;</a>
netopeer	<a href="http://code.google.com/p/netopeer/">&lt;http://code.google.com/p/netopeer/&gt;</a>

# YANG, YIN, XSD, RELAX NG

## 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.

# Built-in Data Types

Category	Types	Restrictions
Integral	{u,}int{8,16,32,64}	range
Decimals	decimal64	range, fraction-digits
String	string	length, pattern
Enumeration	enumeration	enum
Bool and Bits	boolean, bits	
Binary	binary	length
References	leafref	path
References	identityref	base
References	instance-identifier	
Other	empty	

## Type system

The data type system is mostly an extension of the SMIng type system, accommodating XML and XSD requirements.

# Leafs, Leaf-lists, Container, Lists

## leaf

A `leaf` has one value, no children, one instance.

## leaf-list

A `leaf-list` has one value, no children, multiple instances.

## container

A `container` has no value, holds related children, has one instance.

## list

A `list` has no value, holds related children, has multiple instances, has a key property.

**STOP**

```
module jacobs-fake-aaa-module {

    namespace "http://tail-f.com/ns/aaa/1.1";
    prefix aaa;

    organization
        "Jacobs University Bremen";

    contact
        "Juergen Schoenwaelder";

    description
        "This module contains a fake YANG module for some tail-f data
        models and it should only be used for educational purposes.";

    revision 2009-07-30 {
        description "Initial revision.";
    }

    feature ssh-keys {
        description
            "This feature indicate the support of SSH key storage.";
    }

    // ...
}
```



```
container aaa {
  container users {
    list user {
      key "name";

      leaf name {
        type string {
          pattern "[a-zA-Z0-9]+";
        }
        description
          "The name of an account on the system. Note that the name
          root is often associated with special privileges.";
      }
      leaf uid {
        type uint32;
        mandatory true;
        description
          "The id used by the system to identify a user.";
      }
      leaf gid {
        type uint32;
        mandatory true;
        description
          "The id used by the system to identify the user's group.";
      }
    }
  }
}
```

```
leaf password {
    type hashed-password;
    description
        "The hashed password of a user. The special value * means
        no access to the system.";
}
leaf ssh_keydir {
    type string;
    if-feature ssh-keys;
    description
        "The storage location of SSH keys."
}
leaf homedir {
    type string;
    default "/";
    description
        "The home directory of the user.";
}
}
}
```

# CONTINUE

# 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.

# 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?

# Notifications and Operations

## notification

The `notification` statement can be used to define the contents of notifications.

## rpc

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

# YANG Implementations

## Commercial Toolkits (not necessarily complete)

Netconf Central <<http://www.netconfcentral.org/>>  
SNMP Research <<http://www.snmp.com/>>  
Tail-f (ConfD) <<http://www.tail-f.com/>>

## Open Source (not necessarily complete)

jYang <<http://jyang.gforge.inria.fr/jYang/>>  
libsmi <<http://www.ibr.cs.tu-bs.de/projects/libsmi/>>  
pyang <<http://code.google.com/p/pyang>>

## Utilities

yang-mode.el    emacs mode for editing yang

## Acknowledgements

- Martin Björklund, Tail-f
- Phil Shafer, Juniper Networks
- Andy Bierman, Netconf Central
- Shikhar Bhushan (ncclient), Jacobs University
- Siarhei Kuryla (yang for libsmi), Jacobs University
- Ha Manh Tran (netconf testing), Jacobs University

## Disclaimer

All errors on the slides are mine.