YumaPro Quickstart Guide

YANG-Based Unified Modular Automation Tools

Client/Server Quickstart Guide

Version 19.10-12

Table of Contents

1 Preface

1.1 Legal Statements

1.2 Additional Resources

This document assumes you have successfully set up the software as described in the printed document:

YumaPro Installation Guide

Other documentation includes:

YumaPro API Quickstart Guide

YumaPro User Manual

YumaPro netconfd-pro Manual

YumaPro yangcli-pro Manual

YumaPro yangdiff-pro Manual

YumaPro yangdump-pro Manual

YumaPro Developer Manual

YumaPro ypclient-pro Manual

YumaPro yp-system API Guide

YumaPro yp-show API Guide

YumaPro Yocto Linux Quickstart Guide

YumaPro yp-snmp Manual

To obtain additional support you may contact YumaWorks technical support department:

support@yumaworks.com

1.2.1 WEB Sites

- **YumaWorks**
  - https://www.yumaworks.com
  - Offers support, training, and consulting for YumaPro.
- **Netconf Central**
  - http://www.netconfcentral.org/
  - Free information on NETCONF and YANG, tutorials, on-line YANG module validation and documentation database
- **Yang Central**

  - http://www.yang-central.org
  - Free information and tutorials on YANG, free YANG tools for download
- **NETCONF Working Group Wiki Page**
  - http://trac.tools.ietf.org/wg/netconf/trac/wiki
  - Free information on NETCONF standardization activities and NETCONF implementations
- **NETCONF WG Status Page**
  - http://tools.ietf.org/wg/netconf/
  - IETF Internet draft status for NETCONF documents
- **libsmi Home Page**
  - http://www.ibr.cs.tu-bs.de/projects/libsmi/
  - Free tools such as smidump, to convert SMIv2 to YANG

1.2.2 Mailing Lists

- **NETCONF Working Group**
  - https://mailarchive.ietf.org/arch/browse/netconf/
  - Technical issues related to the NETCONF protocol are discussed on the NETCONF WG mailing list. Refer to the instructions on https://www.ietf.org/mailman/listinfo/netconf for joining the mailing list.
- **NETMOD Working Group**
  - https://datatracker.ietf.org/wg/netmod/documents/
  - Technical issues related to the YANG language and YANG data types are discussed on the NETMOD WG mailing list. Refer to the instructions on the WEB page for joining the mailing list.

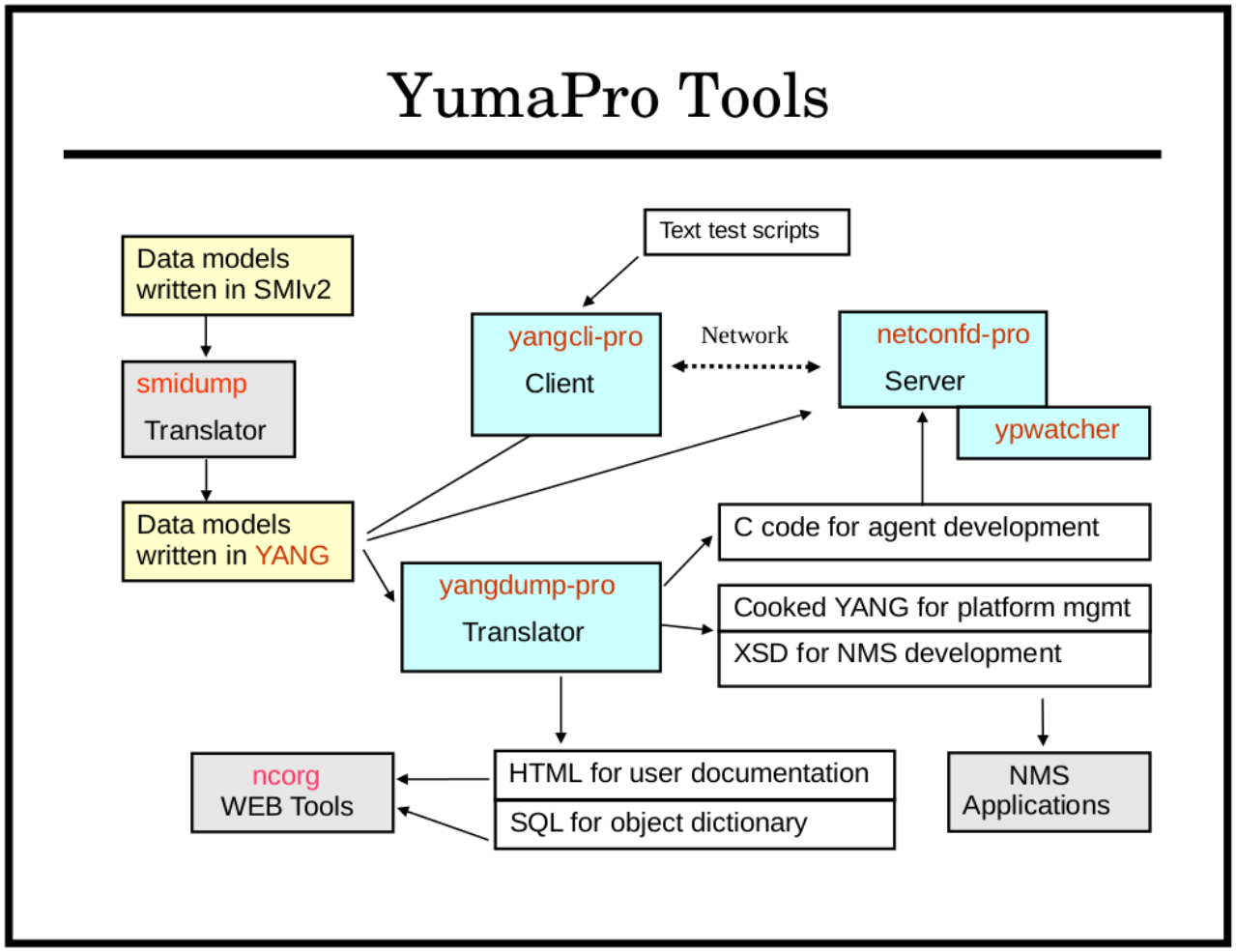
1.3 Conventions Used in this Document

The following formatting conventions are used throughout this document:

Documentation Conventions

Convention	Description
--foo	CLI parameter foo
<foo>	XML parameter foo
foo	**yangcli-pro** command or parameter
$FOO	Environment variable FOO
$$foo	**yangcli-pro** global variable foo
some text	Example command or PDU
some text	Plain text

2 Introduction



Refer to section 3 of the YumaPro User Manual for a complete introduction to YumaPro Tools.

This section focuses on the client and server tools within the YumaPro Tools programs.

2.1 Intended Audience

This document is intended for users of the YumaPro Client and Server package programs. It covers the basic usage of the **yangcli-pro** client application and the **netconfd-pro** server.

2.2 What is NETCONF and YANG?

The YumaPro Tools suite provides automated support for development and usage of network management information. Information is exchanged in XML encoding within a session between a client and a server.

The IETF "Network Configuration Protocol" (NETCONF) is used to provide the management sessions, operations, and database framework available on the server. The operations, notifications, and the database contents supported by a particular NETCONF server are extensible, and defined with a modular and easy-to-learn language called YANG. The

database is used to contain YANG data structures which represent the configuration of the device containing the NETCONF server. This configuration can be saved in non-volatile storage so the configuration can be restored upon reboot.

The IETF "YANG Data Modeling Language" is used to define the syntax and semantics of the NETCONF operations, notification events, and database content. Machine and human readable semantics and constraints are used by YANG tools (including YumaPro Tools) to automate behavior within the NETCONF protocol for clients and servers.

For people familiar with SNMP and SMIv2, NETCONF is like an XML-based, high-level version of SNMP, and a YANG module is like a MIB module, except MIB tables can be nested and much more complex than in SMIv2. Instead of Enterprise IDs and OBJECT-IDENTIFIERs, YANG uses XML namespaces and XPath path expressions to identify module ownership and contents within the protocol PDUs.

2.3 How Does an Operator Use NETCONF and YANG?

An operator uses a NETCONF session almost like it was a CLI session, except there are structured, schema-defined requests and responses, encoded in XML. YANG modules are like MIB modules for CLI content. Instead of ad-hoc unstructured documentation like CLI, NETCONF uses a data definition language to define management modules. The actual modules that a server supports will vary, just like MIB (SMIv2) modules.

The NETCONF protocol is available for many different transports. The most popular is the SSH2 protocol. The 'netconf' subsystem is used (on TCP port 830) to start a special SSH session with the NETCONF server.

Using NETCONF over SSH is just like using CLI over SSH to manage a networking device, except the messages are exchanged in XML, not plain-text. SSH user names and passwords are used for session authentication and authorization.

NETCONF is designed to provide a programmatic interface, so it is usually used with a management application, instead of a direct (raw) SSH terminal application. The **yangcli-pro** program within YumaPro Tools is a YANG-driven NETCONF client application that supports scripts, XPath, and many automated features to simplify management of NETCONF servers.

Once a session is started, similar to a CLI session, the operator issues commands (NETCONF operations) to the server, and the server performs each requested operation in order, and returns a status message and/or some data to the client. Notifications can also be received, if the session has requested them with the <create-subscription> operation.

When a NETCONF session starts, a <hello> message is sent by the server that has all the NETCONF capabilities and YANG modules supported by the server. Capabilities are optional protocol mechanisms, beyond those defined in the base protocol (RFC 4741). The client application knows what operations, notification events, and database contents are supported on the server, based on the information in the <hello> message.

NETCONF has a set of basic database (CRUD) operations for managing the configuration database. In addition, any YANG module can define new protocol operations and notification events.

2.4 How Does a Developer Use NETCONF and YANG?

A NETCONF server developer decides what modules need to be supported by the NETCONF server, and implements the device instrumentation code for those modules.

Much of the NETCONF protocol related code is handled by the NETCONF stack, based on the YANG module contents. Therefore, the most important task for a developer is designing a good YANG module.

After the YANG module is written, the **yangdump** program is used to generate the template C code for the server instrumentation library for the YANG module. The device instrumentation code for the YANG module is then added by the developer. This 'callback code' is called from the NETCONF stack when database operation requests for the object(s) in the YANG module are received by the server.

Once this library is completed, the YANG module and its binary server instrumentation library (SIL) can be loaded into the NETCONF server at run-time. There is no need to recompile the **netconfd-pro** server, or even reboot it.

Once the YANG module and its instrumentation code is completed, it needs be published, so operators and application developers can use the WEB documentation and other tools to access the information in the NETCONF server.

3 Getting Started with toaster.yang

This section will demonstrate the basic operation of YumaPro Tools to use a NETCONF session to manage a remote device with a YANG data model. The YumaPro Tools programs and libraries must already be installed. Refer to the YumaPro Tools Installation Guide if this has not yet been done.

The **yangcli-pro** client program and **netconfd-pro** server program do not need to be installed on the same machine. Usually, they are not installed on the same machine. For simplicity, the server address 'localhost' is used in the examples below.

Libtoaster Fast Start:

This command summary is explained in depth in the rest of this section.

1. Start the server with your user as the superuser
    1. /usr/sbin/netconfd-pro –superuser=<username>
2. Start yangcli-pro
    1. /usr/bin/yangcli-pro
3. yangcli-pro commands to the server
    1. connect server=localhost user=<username> password=<password>
    2. create-subscription
    3. load toaster
    4. mgrload toaster
    5. make-toast (will fail)
    6. get-locks
    7. create /toaster
    8. save
    9. release-locks
    10. sget /toaster
    11. xget /toaster
    12. make-toast toasterDoneness=4 toasterToastType=toast:frozen-waffle
    13. cancel-toast
    14. close-session

3.1 What is libtoaster?

There is a sample server instrumentation library (SIL) included, named libtoaster. This is the module-specific server instrumentation code for the management data defined in toaster.yang. This is based on the original TOASTER-MIB by Epilogue. This YANG module provides simple operations to make toast, and some simple NETCONF database objects to enable and monitor the toaster.

The new YANG version of the TOASTER-MIB is different is some ways:

- extensible YANG identities are use to identify the bread type, instead of a hard-wired enumerated list.
- protocol operations (<make-toast> and <cancel-toast>) are instead of an 'up/down' switch within the database. NETCONF databases are intended to contain persistent data structures, and 'actions' such as starting or stopping the toaster are done with new protocol operations, instead of editing the database with the standard operations.
- A simple configuration 'presence container' object is used to enable and disable the toaster service, instead of hard-wiring the toaster service availability.
- A notification is generated when the toast is done or canceled. This notification can be used instead of polling the toaster status object.

The complete text of **toaster.yang** and TOASTER-MIB can be found at the end of this document.

3.2 Start the netconfd-pro server

If the **netconfd-pro** server is already running, then skip this section.

Details for all the **netconfd-pro** configuration parameters can be found in the YumaPro Netconfd-pro Manual.

There is also a sample configuration file (default location):

- **/etc/yumapro/netconfd-pro-sample.conf**

This file contains all the default settings for the configuration parameters.

To change the default settings, copy and edit this file (example commands)

```
mydir> sudo cp /etc/yumapro/netconfd-pro-sample.conf \
          /etc/yumapro/netconfd-pro.conf
mydir> sudo emacs /etc/yumapro/netconfd-pro.conf
```

3.2.1 Configuration Defaults

To keep the example simple, the default settings will be used:

- the server will accept sessions on TCP port 830
- the target database is <candidate>
- no <startup> database (mirrored NV-save)
- the <validate> operation is supported
- the access control mode is 'enforcing'
- the super user account name is 'superuser'
- the server will search startup-cfg.xml using the default search path
- the default <with-defaults> behavior is 'explicit'
- notification replay is enabled with a buffer size of 1000 events and a maximum message burst per session of 10 notifications
- the <hello> exchange timeout is 10 minutes
- the session idle timeout is 1 hour
- the default session indent amount is 2 spaces
- the default session line-size is 72 characters
- violation of strict YANG XML ordering will not cause errors
- logging level 'info' is enabled and sent to STDOUT
- the default watcher-interval is 10 seconds

3.2.2 SSH Server

To start the NETCONF server, make sure that the **sshd** server is running, and the following configuration is included in **/etc/ssh/sshd_config.**

```
Port 22
Port 830
Subsystem netconf /usr/sbin/netconf-subsystem-pro
```

The 'Subsystem' command may be different if **netconf-subsystem-pro** has been installed in a different location than **/usr/sbin**. The 'Port 22' command is needed to make sure the SSH server will accept SSH sessions in addition to NETCONF sessions.

3.2.3 NETCONF Server

The current working directory in use when **netconfd-pro** is invoked is important. It is most convenient to run **netconfd-pro** in the background, and save all output to a log file.

The **netconfd-pro** program listens for connection requests on a local socket, that is located in **/tmp/ncxserver.sock**.

In order for NETCONF sessions to be enabled, the **SSH server** and the **netconf-subsystem** programs must be properly installed first.

The **netconfd-pro** program does not directly process the SSH protocol messages. Instead, it is implemented as an SSH subsystem. The number of concurrent SSH sessions that can connect to the **netconfd-pro** server at once may be limited by the SSH server configuration. Refer to the YumaPro Installation Guide for more details on configuring the SSH server for use with **netconfd-pro**.

The **netconfd-pro** program can be invoked several ways:

- To get the current version and exit:

```
netconfd-pro --version
```

- To get program help and exit:

```
netconfd-pro --help
netconfd-pro --help --brief
netconfd-pro --help --full
```

- To start the server in the background, set the logging level to 'debug', and send logging messages to a log file

```
netconfd-pro --log-level=debug --log=~/mylog &
```

- To start the server interactively, and send all log messages to STDOUT:

```
netconfd-pro
```

- To start the server interactively, with a new log file:

```
netconfd-pro --log=mylogfile
```

- To start the server interactively, and append to an existing log file:

```
netconfd-pro --log=mylogfile –log-append
```

- To get parameters from a configuration file:

```
netconfd-pro –config=/etc/yumapro/netconfd-pro.conf
```

3.2.4 Stopping netconfd-pro

To terminate the **netconfd-pro** program when running interactively, use the control-C character sequence. This will cause the server to cleanup and terminate gracefully.

The <shutdown> or <restart> operations can also be used to terminate or restart the server. The **yuma-nacm.yang** access control rules must be configured to allow any user except the 'superuser' account to invoke this operation.

3.2.5 Starting netconfd-pro with ypwatcher program

The **ypwatcher** is a program that provides monitoring mechanism to **netconfd-pro** server and its state. **Ypwatcher Ypwatcher** program periodically checks the server's state and determine if the server is still running. If the server is no longer running it cleans up the state, restarts the server, and generates a syslog message.

The **ypwatcher** program will be launched by the server by default unless **–no-watcher** parameter will be specified or the program is already running.

The **ypwatcher** program is running continuously and attempting to restart the server any time it exits unexpectedly.

The **ypwatcher** program will be invoked automatically whether the server starts interactively or in the background mode:

- To start the server interactively, with **ypwatcher** program:

```
netconfd-pro
```

- To start the server interactively, with no **ypwatcher** program:

```
netconfd-pro --no-watcher
```

The **–watcher-interval** parameter specifies the sleep interval between **ypwatcher** program attempts to check availability of the server.

- To start the server interactively, with **ypwatcher** program and set the watcher interval:

```
netconfd-pro --watcher-interval=10
```

3.3 Start the yangcli-pro client

Once the NETCONF server is running, it will accept client sessions If running **netconfd-pro** interactively on localhost, then start a new terminal window to continue.

The **yangcli-pro** program should be in found in the PATH environment variable.

The **yangcli-pro** program can be invoked several ways:

- To start the client interactively, and send all log messages to STDOUT:

```
yangcli-pro
```

- To get the current version and exit:

```
yangcli-pro --version
```

- To get program help and exit:

```
yangcli-pro --help
yangcli-pro --help --brief
yangcli-pro --help --full
```

- To set the logging level to 'debug':

```
yangcli-pro --log-level=debug
```

3.3.1 Configuration Defaults

To keep the example simple, the default settings will be used:

- the client will attempt to start sessions on TCP port 830
- the client will attempt to automatically complete partial commands
- the command line history will be automatically loaded upon startup, and saved upon exit
- the client will attempt to automatically load any YANG modules advertised in the server <hello> message
- the client will check before using invalid parameter values
- the plain display mode will be used, with 72 characters per line
- each nest level of displayed data will be indented 2 spaces
- the XML order of messages sent to the server will be corrected, as needed
- the logging level of 'info' is set, and log messages are sent to STDOUT
- the client will wait 30 seconds for responses

3.3.2 Startup Screen

The startup screen shows the following information:

- program version and copyright
- tab key can be used for command and parameter completion
- basic help instructions
- basic statement instructions

3.3.3 Command Line Editing

The command lines are stored in a history buffer.

Any previous command line (except a password parameter line) can be recalled and used again.

Any command in the command buffer (current or recalled) can be edited. The default key settings are aligned with the emacs editor. Refer to the YumaPro Tools User Manual for more details.

3.3.4 Escape Commands

Not all parameters need to be entered at one time. If yangcli-pro needs more information, based on the initial command line, then 1 or more missing parameters will be requested, in sequence.

It is possible to get help, skip a parameter, or even cancel the entire command during one of these sub-command modes, by using an escape command. This is a 1 or 2 character command, followed by the 'enter' key (as usual to end a command).

Escape Command Summary

command	description
?s	skip the current parameter
?c	cancel the current command
?	get help
??	get full help

Using the '?s' command to skip a parameter may cause the <rpc> request to be invalid.

Depending on the setting of the **--bad-data** configuration parameter, this may or may not be allowed. The default setting is to warn and confirm. This configuration parameter also affects parameter values that are invalid according to the YANG module definition.

3.4 Getting Context Sensitive Help

The **yangcli-pro** program provides context-sensitive help based on the current NETCONF session status and the set of YANG modules currently loaded.

When a NETCONF session is active, the set of modules advertised in the <hello> message by the server will be used to generate help text, if available. The '**mgrload**' command can be used to force **yangcli-pro** to use different or additional YANG modules.

If the **yangcli-pro** program does not have the advertised revision of a particular module available in the module search path, and the NETCONF server supports the standard <get-schema> operation, then the module will be retrieved from the server, and used just for that session.

If any features or deviations are advertised for a YANG module, then they will be applied to the modules used just for the current session. The help text and the error checking done for the module will be based on this 'patched' module, not the 'plain' module specified in the capability URI string.

3.4.1 Tab Key for Command Completion

The 'tab' key can be used at any time to see a list of the possible completions that the command interpreter will accept. The list will be displayed for command names and some command parameters.

When a NETCONF session is active, all the NETCONF operations will be available. Additional commands may also be available if the server advertised any YANG modules containing 'rpc' statements.

3.4.2 The '?' Escape Sequence in Command Mode

When entering input in command mode, if a partial command is entered, or if a data structure is being filled, then the context-sensitive help escape character ('?') is available to get help about that command, parameter or data node. Use one question mark for to access context-sensitive help, similar to using the tab key.

3.4.3 The '?' and '??' Escape Sequences in Parameter Mode

When entering input in parameter mode, if a partial command is entered, or if a data structure is being filled, then the help escape sequences are available to get help about that parameter or data node. Use one question mark for help, and two question marks for maximum help.

Help Escape Sequences

sequence	description
?	Print some help text, but not description statements and some other information.
??	Print maximum help text.

The following example shows the help text for the 'user' parameter for the 'connect' operation:

```
yangcli-pro> connect

Enter string value for leaf <user>
yangcli-pro:connect> ?

   leaf user [NcxUserName]
      length: 1..63
      pattern: [a-z,A-Z][a-z,A-Z,0-9,\-,_,\.]{0,62}

Enter string value for leaf <user>
yangcli-pro:connect>
```

The type of object, its name, data type, and any restrictions, will be printed.

After that, the previous prompt will be redisplayed.

3.4.4 The 'help' Command

The **help** command can be used to display all kinds of information about the **yangcli-pro** program and the YANG data module contents in use at the time.

Help Command Variants

command	description
help <comand-name> help command <command-name>	Display help for the specified yangcli-pro command or YANG rpc statement.
help commands	Display help text for all commands.
help object <object-name>	Display help text for a YANG database top-level object (only if its module is available).
help notification <notification-name>	Display help text for a YANG notification event (only if its module is available).
help type <type-name>	Display help text for an exported YANG data type (only if its module is available).

Each of the help command variants also accepts a 'help-mode' parameter to control how much help text is displayed:

Help Output Modes

mode	description
--brief	Display minimal help text.
--normal	Display a lot, but not always all the help text available (default mode).
--full	Display all available help text, including description statements.

The following table shows some valid help commands:

command	description
help help	Get normal help for the help command.
help commands brief	Get a 1 line description of each command.
help object system full	Get all available help for the /system container and all its descendant nodes.
help type NcxIdentifier	Get summary and description of the data type called 'NcxIdentifier'.
help notification sysSessionStart	Get a summary of the 'sysSessionStart' notification, and each of objects in its payload.

3.5 Start a NETCONF session

Each yangcli-pro program instance can run 1 NETCONF session at a time.

If no session is currently active, then the prompt will contain just the program name, indicating that the 'connect' command is available:

```
yangcli-pro>
```

3.5.1 The connect Command

The 'connect' command is used to start a NETCONF session.

There are 3 mandatory parameters for this command:

- **user**: the system (or SSH) user name to use
- **server**: the IP address or DNS name of the NETCONF server to use
- **password**: the password string to use

Make sure you have a user name and password already configured on the NETCONF server.

If a partial command is given, then yangcli-pro will prompt for any missing mandatory parameters. In this example, the complete command is given at once:

```
yangcli-pro> connect server=localhost user=guest password=yangrocks
```

After this command is entered, **yangcli-pro** will generate some informational log messages to the screen.

If the session is started successfully, a summary of the server session capabilities and available modules should be displayed. Also, the command prompt will change to indicate that a NETCONF session is currently active.

```
guest@localhost>
```

At this point any command supported by the server can be entered, in addition to any **yangcli-pro** command (except 'connect').

3.5.2 Fixing Connection Problems

If the session did not start correctly, check the error messages to fix the problem. Some common problems:

- Make sure the **netconfd-pro** program is running.
- Make sure the **netconf-subsystem-pro** program is properly installed.
- Check if the SSH configuration contains the portion for NETCONF.
- If the SSH configuration looks correct, then try restarting the SSH server to make sure that configuration file is the one being used.
- If the SSH server seems to be running correctly, then check if any firewall or other security mechanism is blocking TCP port 830. If so, either enable TCP port 830, or enable port 22 on the NETCONF server (by restarting the server), and include 'port=22' in the 'connect' command parameters.
- If no firewall or other security measure is blocking TCP port 830, try to establish a normal SSH session with the server.
- If a normal SSH session works correctly, then check the log messages on the NETCONF server for more information.

3.6 Enable Notification Delivery

In order to receive the 'toastDone' notification event, a notification subscription has to be enabled.

A default NETCONF notification stream can be started with the 'create-subscription' command:

```
guest@localhost> create-subscription

RPC OK Reply 2 for session 1:

guest@localhost>
```

Depending on other activity within the NETCONF server, it is possible other notification events, such as 'sysSessionStart' or 'sysSessionEnd' will be generated. Notifications are displayed in their entirety, but not during 'rpc reply output'. If a command is being entered, the notification will be displayed, and then the command line restored.

3.7 Load the Toaster Module

The toaster module is not a core system module, and is not available automatically.

The module has to be explicitly loaded by the NETCONF client.

To load the server-supported version of the toaster module, use the 'load' command:

```
guest@localhost> load toaster

RPC Data Reply 2 for session 1:

rpc-reply {
   mod-revision 2009-11-20
}

Incoming notification:
   notification {
      eventTime 2009-12-28T00:44:45Z
      sysCapabilityChange {
         changed-by {
            userName guest
            sessionId 1
            remoteHost 127.0.0.1
         }
         added-capability
          http://netconfcentral.com/ns/toaster?module=toaster&revision=2009-11-20
      }
      sequence-id 3
   }

guest@localhost>
```

If the module was successfully loaded, then a data response will be sent, containing the revision date of the toaster module that was loaded. This response will be returned even if the module was already loaded.

Note that the 'sysCapabilityChange' notification event will only be sent if the module has not already been loaded into the server. In this case, it was not advertised in the <hello> message for this session, and the toaster module needs to be loaded manually into yangcli-pro with the 'mgrload' command:

```
guest@localhost> mgrload toaster

Load module 'toaster' OK

guest@localhost>
```

3.8 Enable the Toaster

Try to make some toast, using the 'make-toast' command:

```
guest@localhost> make-toast

RPC Error Reply 4 for session 1:

rpc-reply {
   rpc-error {
      error-type protocol
      error-tag resource-denied
      error-severity error
      error-app-tag no-access
      error-message 'resource denied'
      error-info {
         error-number 269
      }
   }
}

guest@localhost>
```

What happened?

A 'resource-denied' error was returned instead of 'OK', because the toaster service is not enabled yet. A node has to be created in the NETCONF database before the 'make-toast' command can be used.

3.8.1 Lock the Databases

The first step is to lock the NETCONF databases for writing. Locks do not affect read operations.

The yangcli-pro program has a high-level command to deal with locking, called 'get-locks'. It will handle retries for any missing locks, until an overall timeout occurs or all the locks needed are acquired.

```
guest@localhost> get-locks

Sending <lock> operations for get-locks...

get-locks finished OK
guest@localhost>
```

3.8.2 Create the toaster Container

The toaster module uses a simple YANG 'presence' container to configure the toaster service.

Once the /toaster container is created, the read-only nodes within that container will be maintained by the server, and the toaster service will be enabled.

The first step is to create the /toaster node in the <candidate> configuration database:

```
guest@localhost> create /toaster

Filling container /toaster:
RPC OK Reply 5 for session 1:

guest@localhost>
```

Now the /toaster node is created in the <candidate> database.

3.8.3 Save the Database Changes

In order to activate these changes, the 'save' command needs to be issued.

This is a high-level **yangcli-pro** operation that issues the 'commit' or 'copy-config' operation, depending on the database target for the current session.

```
guest@localhost> save

Saving configuration to non-volative storage
RPC OK Reply 6 for session 1:


Incoming notification:
   notification {
      eventTime 2009-12-28T00:59:58Z
      sysConfigChange {
         userName guest
         sessionId 1
         remoteHost 127.0.0.1
         edit {
            target /toast:toaster
            operation create
         }
      }
      sequence-id 4
   }
```

The 'RPC OK' message indicate that the server successfully saved the configuration.

The 'sysConfigChange' notification indicates what was changed in the running configuration, and who made the change(s).

The toaster server should now be enabled.

3.8.4 Unlock the Databases

The database locks need to be released as soon as possible after the edits are completed or discarded.

The high-level command 'release-locks' must be used if 'get-locks' was used to acquire the database locks.

```
guest@localhost> release-locks

Sending <unlock> operations for release-locks...

guest@localhost>
```

3.9 Get the Toaster State Information

To discover the toaster model and its current status, the 'sget' or 'xget' commands can be used to retrieve just the toaster portion of the conceptual state data available on the server.

The 'sget' command is high-level subtree filter handler for the <get> operation:

```
guest@localhost> sget /toaster


rpc-reply {
  data {
    toaster {
      toasterManufacturer 'Acme, Inc.'
      toasterModelNumber 'Super Toastamatic 2000'
      toasterStatus up
    }
  }
}
```

The 'xget' command is high-level XPath filter handler for the <get> operation. It is only available if the NETCONF server supports the **:xpath** capability (like **netconfd-pro**).

```
guest@localhost> xget /toaster

rpc-reply {
  data {
    toaster {
      toasterManufacturer 'Acme, Inc.'
      toasterModelNumber 'Super Toastamatic 2000'
      toasterStatus up
    }
  }
}
```

Both commands should return the same data:

```
Filling container /toaster:
RPC Data Reply 7 for session 1:

rpc-reply {
   data {
      toaster {
         toasterManufacturer 'Acme, Inc.'
         toasterModelNumber 'Super Toastamatic 2000'
         toasterStatus up
```

```
      }
   }
}
```

This data shows that the 'Super Toastamatic 2000' is ready to make toast!

3.10 Start Making Toast

Now that the toaster is enabled, the 'make-toast' command should work.

Instead of using the default parameter values, let's make a frozen waffle a little less done than normal:

```
guest@localhost> make-toast toasterDoneness=4 toasterToastType=toast:frozen-waffle

RPC OK Reply 8 for session 1:
```

At this point the toaster timer is running, and the simulated waffle is cooking,

After about 40 seconds, the 'toastDone' notification should be received:

```
Incoming notification:
   notification {
      eventTime 2009-12-29T01:20:05Z
      toastDone {
         toastStatus done
      }
      sequence-id 5
   }
```

This 'toastDone' event shows that the toast was completed, and is ready to eat.

3.11 Stop Making Toast

What if you change your mind, and want wheat toast instead of a waffle?

Repeat the previous command (Control-P should recall the previous command):

```
guest@localhost> make-toast toasterDoneness=4 toasterToastType=toast:frozen-waffle

RPC OK Reply 9 for session 1:
```

Now enter the 'cancel-toast' command right away, before the waffle finishes:

```
guest@localhost> cancel-toast

RPC OK Reply 10 for session 1:

Incoming notification:
   notification {
      eventTime 2009-12-29T01:24:36Z
      toastDone {
         toastStatus cancelled
      }
      sequence-id 6
   }
```

This 'toastDone' event shows that the toast was cancelled.

3.12 Close the NETCONF Session

To close the NETCONF session, use the 'close-session' command:

```
guest@localhost> close-session

RPC OK Reply 11 for session 1:

ses: session 1 shut by remote peer
yangcli-pro>
```

Note that the prompt returned to the default form, once the session was dropped by the NETCONF server.

The terminate the yangcli-pro program, use the 'quit' command:

```
yangcli-pro> quit

mydir>
```

4 Advanced Topics

This section introduces some advanced features of the NETCONF protocol and YANG data modeling language.

4.1 Data Retrieval

4.1.1 Basic NETCONF Retrieval Operations

The NETCONF protocol has 2 different retrieval operations:

- **<get>**: get state data and the running configuration database.
- **<get-config>**: get just the specified configuration database.

Each of these operations accepts a <filter> parameter, which has 2 forms:

- **subtree filter:** retrieve just the subtrees in the database that match the XML subtrees in the filter.
- **XPath filter:** retrieve just the subtrees that match the result node set produced by evaluating the specified XPath expression against the database. This mode cannot be used unless the **:xpath** capability must be advertised by the server.

The **yangcli-pro** program supports 3 different forms of each command:

- **plain**: plain NETCONF operation with user-supplied filter
- **subtree**: XPath path expression or user variable is converted to XML for the <filter> parameter subtree XML.
- **xpath**: XPath path expression or user variable is converted to XML for the <filter> parameter 'select' XML attribute

yangcli-pro Retrieval Commands

command	description	example
get	plain <get> operation	get with-defaults=trim
get-config	plain <get-config> operation	get-config source=candidate
sget	<get> with a subtree filter	sget /system
sget-config	<get-config> with a subtree filter	sget-config source=running /nacm/rules
xget	<get> with an XPath filter	xget "//interface/counters"
xget-config	<get-config> with an XPath filter	xget-config source=candidate "//interface[name='eth0']"

The retrieval commands return an element named <data> containing the requested XML subtrees.

If any identifier nodes (YANG key leafs) are needed to distinguish the data in the reply, they will be added as needed by the server. In the 'xget' example above, the <name> element for each interface would be returned, even though it was not directly requested by the XPath expression.

4.1.2 Default Value Filtering

The data will also be filtered according to the defaults handling behavior of the server, unless the <with-defaults> parameter is added to the command. This parameter is only supported if the server advertised the 'with-defaults' capability, If not, the client does not get any indication from the server what type of defaults filtering is being done (if any).

There are 4 types of defaults filtering provided:

- **report-all-tagged**: no filtering -- return all nodes even those the server might normally suppress because they are considerer to be default values by the server. Add the 'default' XML attribute to nodes that were set by default.
- **report-all**: no filtering -- return all nodes even those the server might normally suppress because they are considerer to be default values by the server.
- **trim**: return all nodes except skip any leaf nodes that match the schema defined default value
- **explicit**: return all nodes that were set by the client or the server to some value, even if the value happens to be the schema defined default. This is normally the default behavior for the **netconfd-pro** server.

The defaults handling behavior can be changed just for a specific NETCONF session, using the <set-my-session> operation. This is only available on the **netconfd-pro** server.

```
guest@localhost> set-my-session with-defaults=report-all

RPC OK Reply 12 for session 1:

guest@localhost>
```

In this example, the 'basic' behavior is changed from 'explicit' to 'report-all', but just for session 1. This setting is temporary, and will not be remembered when the session is terminated. If the <with-defaults> parameter is present, it will be used instead of this value.

4.1.3 Special Retrieval Operations

Any YANG module can add new operations with the 'rpc' statement.

New retrieval operations may also be added which are associated with a protocol capability.

Just like any other data model content, the operator (or application) needs to understand the YANG file definitions, including the description statements, to understand how each custom retrieval operation works.

There are 2 custom retrieval operations supported by **netconfd-pro**:

Special Retrieval Operations

operation	description
get-schema	Retrieve the YANG or YIN source file for one of the modules advertised by the server. This is a standard operation defined in the **ietf-netconf-monitoring** module.
get-my-session	Retrieve the customizable settings for my session. This is a proprietary operation defined in the **yuma-my-session** module.

4.2 Notifications

Notifications are used in NETCONF to send server event information to the client application.

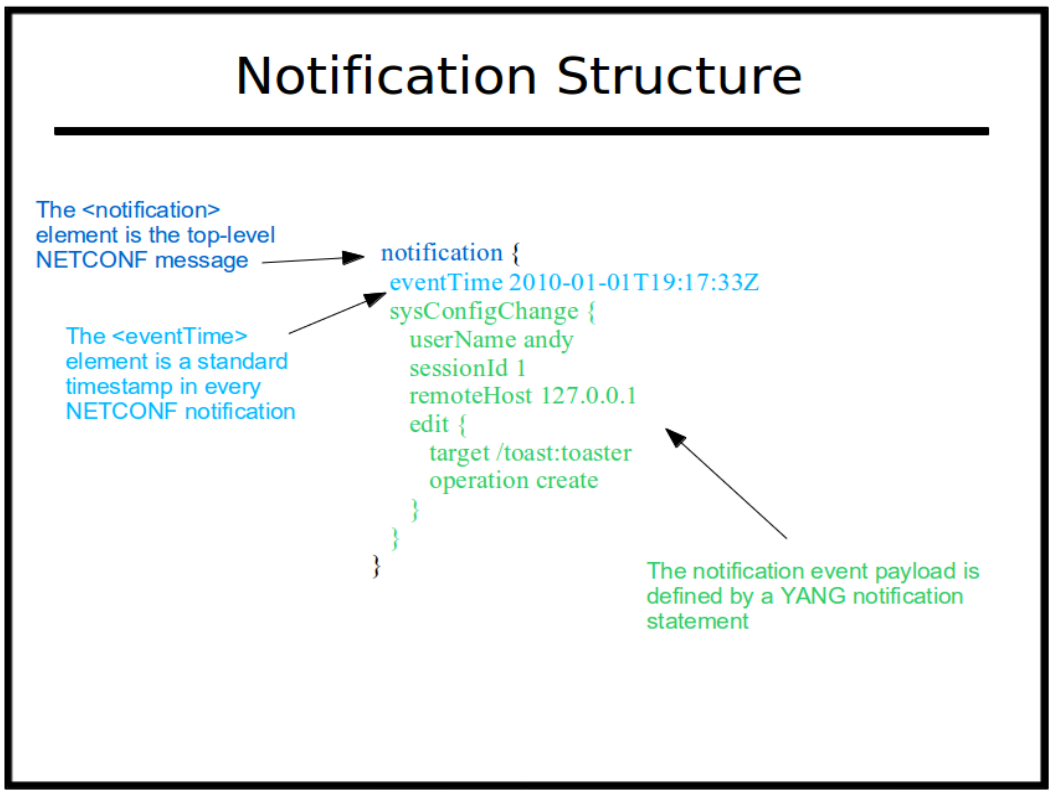
A session must request notifications with the 'create-subscription' command.

Notifications are grouped into 'streams', but only the 'NETCONF' stream is defined at this time.

A notification subscription request specifies the stream name (and perhaps more parameters).

A NETCONF session on the **netconfd-pro** server will never expire due to inactivity, while a notification subscription is active. This allows notification processing applications to maintain long-lived connections without worrying about a NETCONF timeout. Note that the SSH server may also be configured to drop idle SSH sessions, whether a notification subscription is active or not.

4.2.1 Notification Contents



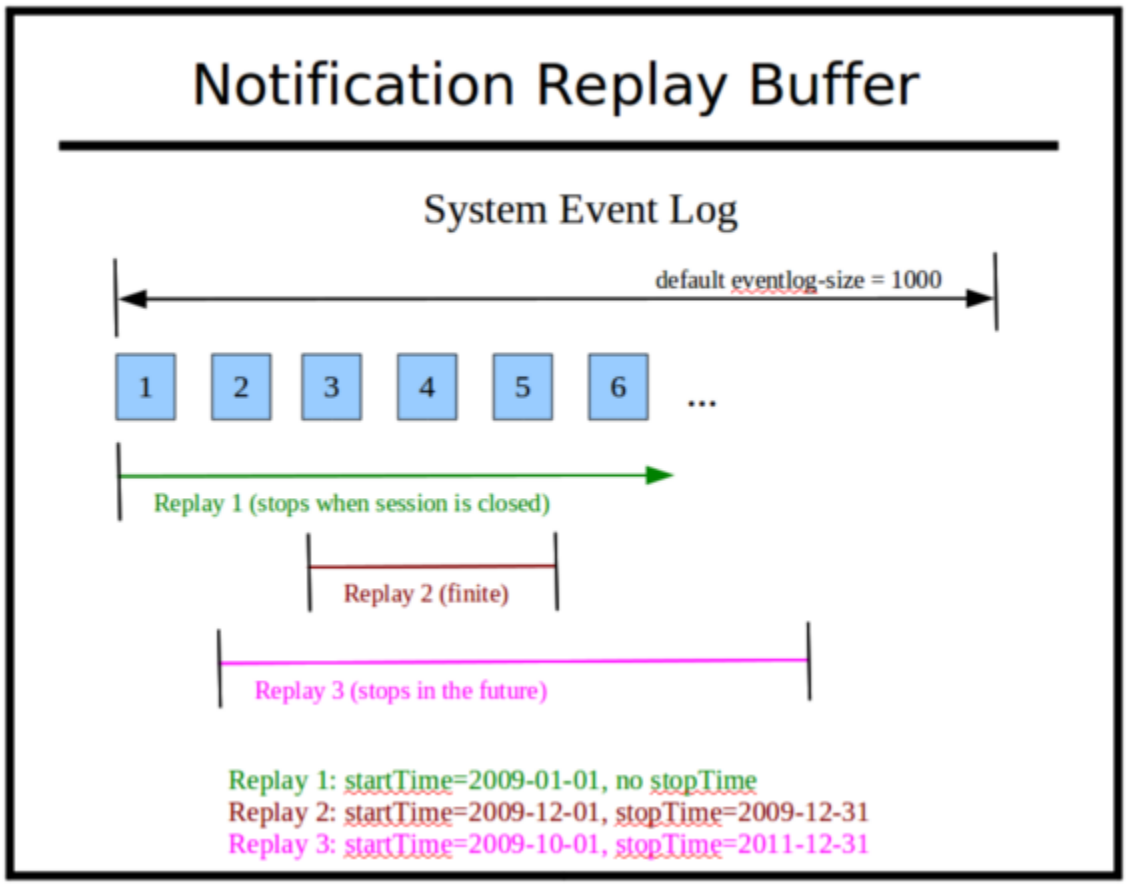
The 'notification' element is sent from the server to the client, if an event occurs, and the client has created a notification subscription.

The child nodes of this element comprise the notification content, and it is divided into 3 sections:

1. **event generation time-stamp**: This standard NETCONF leaf is always the first child element within the notification element.

2. **event payload**: The module-specific event payload is represented as a container with the name of the notification. Any data nodes defined within the YANG notification statement appear (in order) as child nodes of the event type container.
3. **proprietary extensions**: Zero or more vendor-specific elements may appear after the event payload element. For example, the monotonically increasing 'sequence-id' element is added to each notification saved in the **netconfd-pro** event log.

4.2.2 Notification Replay



The NETCONF server will maintain an ordered buffer of saved notification events, if the :notification-replay capability is supported by the server. For the **netconfd-pro** server, this is a configurable feature, set by the **--eventlog-size** parameter.

The **netconfd-pro** default is to save the most recent 1000 notification events.

Only system events are saved and are available for retrieval. The 'replayComplate' and 'subscriptionComplete' events are session-specific events, and are therefore not saved in the replay buffer.

The 'create-subscription' command has 2 parameters to request that stored notifications be delivered to the client session:

- **startTime**: the date (or date-and-time) to compare against the event generation time-stamp. Only notification events that occurred after this time are delivered.
- **stopTime**: the date (or date-and-time) to compare against the event generation time-stamp. Only notification events that occurred before this time are delivered. This parameter can specify a time in the future. When that time has passed, the subscription will be terminated. The stopTime does not cause the server to wait that period of

time to generate an event. If the stopTime is in the past, then the subscription will terminate after all the matching event timestamps in the replay buffer have been delivered.

Notifications are delivered in the order they are stored. Each new **netconfd-pro** notification contains a monotonically increasing sequence-id (unsigned integer). This can be used to help determine if any configured notification filters are working as expected.

4.2.3 The interleave capability

The **netconfd-pro** server supports the :interleave capability, which means that all commands (except create-subscription) will be accepted by the server. The client should expect <rpc-reply> and <notification> messages. The server will always maintain proper message serialization. These messages will always be sent in their entirety, which may impact applications (e.g., a really long <get> response on the same session will delay notification delivery).

If the NETCONF server does not support the :interleave capability, then it may only allow the <close-session> operation while the notification subscription is active. In this case, a new NETCONF session is required to perform any management operations.

This special mode is only applicable while a notification subscription is active. It is possible for a replay subscription to terminate, without terminating the session as well. In this case, the 'notificationComplete' event will be generated, and the session will return to accepting all possible operations.

4.3 Database Editing

NETCONF supports multiple conceptual configuration databases. Only the 'running' database is actually active. All other databases are scratch-pad databases, or some other special-purpose off-line database.

Every NETCONF server must allow arbitrary partial (and concurrent) editing to its configuration with the <edit-config> operation. Refer to the YumaPro Tools User Manual for complete details on this NETCONF operation. The **yangcli-pro** program has simplified editing commands, which are explained below.

The <config> element within an <edit-config> PDU represents the 'root node' (/) in the path expression for each node in the conceptual database. Each top-level YANG object that is supported and configured will be represented as child nodes this root node. The conceptual database can be processed as an XML instance document with multiple top nodes (similar to XSLT rules).

Database editing in NETCONF has several variants, but basically, it follows this simple procedure:

1. Lock the database(s) that will be affected.
2. Use <edit-config> or <copy-config> on the target database to make changes.
3. Activate and save the database edits.
4. Unlock the database(s) that were previously locked.

4.3.1 The Target Database

Usually a NETCONF server supports the <edit-config> operation on only one database, which is either the candidate or the running database. This is called the 'target' database, which corresponds to the <target> parameter in the <edit-config> operation.

If the target database is the candidate configuration, then the <edit-config> operation does not always cause all possible database validation checking to be done by the server. Since the candidate database is just a scratch-pad for (possibly) incremental edits, the server is not required to completely validate its contents. Instead, these 'final validation' tests are only required to be done when the <commit> operation is invoked.

The **yangcli-pro** program will automatically handle the target database management, based on the server capabilities reported each session, if the 'save' command is used. The manual procedure (<commit> and/or maybe <copy-config> operations) is also supported, but do not mix them within the same editing session.

4.3.2 Database Locking

NETCONF supports database locking so a session can have exclusive write access to the configuration.

These locks are intended to be short-lived, but there is no actual time limit on a lock. If the session terminates for any reason with any locks, they will be released automatically by the server.

All the databases that are involved in the edit should be locked. This always includes the running database, and the candidate and startup databases, if they are supported by the server.

The yangcli-pro program has 2 special commands to handle all locking:

- **get-locks**: Wait until all database locks have been acquired or the timeout occurs
- **release-locks**: Rlease any locks that were obtained with get-locks

Refer to the YumaPro Tools User Manual for more details on these commands.

4.3.3 Non-Volatile Storage

The startup configuration is the conceptual database used on the next reboot of the NETCONF server. It is important to know whether the NETCONF server supports the :startup capability or not. If yes, then the operator must explicitly save the running database to non-volatile storage (the startup database), using the <copy-config> operation. If no, then the server will keep the running and startup databases synchronized.

The **yangcli-pro** program has a high-level 'save' command, used after the editing operations, that will automatically issue the correct protocol operations to complete the edit, and save the changes in non-volatile storage.

The startup database is configurable in the **netconfd-pro** server. The **--with-startup** configuration parameter controls whether the startup database will be used or not. The --startup parameter can be used to control the initial load of the running configuration in 3 different ways:

1. **no startup**: skip this step and just use factory defaults
2. **default startup**: look for the default **startup-cfg.xml** file in the configured data path.
3. **specific startup**: use a specified file, either absolute file-spec, or a relative path in the configured data path

Refer to the YumaPro Tools User Manual for more details on controlling non-volatile storage.

4.3.4 Editing Commands

The <edit-config> operation should be used make configuration changes. The <copy-config> operation can also be used, but this is a blunt hammer approach. Although the **netconfd-pro** server will always analyze the edit request and only affect the nodes that actually changed, this is not a requirement in the standard.

The <edit-config> operation allows the operator to have precise control of the server. These database edits are performed by the server using a combination of 3 factors:

1. The nodes that currently exist in the target database.
2. The nodes that exist in the 'source' of the edits (either the inline <config> element or indirectly through the <url> element.
3. The <default-operation> parameter and any XML attributes in the source XML elements (nc:operation attribute and YANG insert operation attributes).

The **yangcli-pro** program provides some high-level commands to automatically handle the complexity of the <edit-config> operation. These commands use XPath expressions and a series of interactive prompts (e.g., for the mandatory nodes and key leafs) to fill in the specified data structures, and construct an optimized NETCONF message.

yangcli-pro Editing Commands

command	description
create	Create a new sub-tree, only if it does not already exist
delete	Delete an existing sub-tree, only if it exists
merge	Merge the source sub-tree into the target sub-tree, keeping any existing nodes that are not explicitly contained in the source.
replace	Merge the source sub-tree into the target sub-tree, deleting any existing nodes that are not explicitly contained in the source. This is the mode used for the <copy-config> operation.

command	description
insert	Insert or move a YANG list or leaf-list entry

Refer to the YumaPro Tools User Manual for details on these commands.

4.4 Access Control

The **netconfd-pro** server can be configured to give precise access rights to each user (the SSH user name associated with the NETCONF session). By default, the access control model used is 'ietf' (RFC 6536).

Some important points to remember about access control:

- There are 3 types of access -- read, write, and execute.
- If a user does not have read access to some data, then it is silently omitted from the reply.
- The 'access-denied' error is not generated for read requests. It is only generated for write requests to the database, or <rpc> operation execution requests.
- An access request results in 1 of 2 outcomes: permit or deny
- The server resolves the access request by searching the access control rules. Either an explicit rule will apply, or the default access rights will be checked if no rule is found.
- The default access rights are configurable, but usually set as follows:
  - read access is permitted
  - write access is denied
  - exec access is permitted
- The **nacm:default-deny-write** and **nacm:default-deny-all** extensions can be used by the YANG module author to override the default access rights, and deny access instead. For example, the <reboot> operation is not permitted by default.
- There is a configurable 'superuser' user name. If desired, a specific user name will be considered the 'super user' and all access control will be bypassed for this user (e.g., **--superuser=joe**). By default, there is no superuser account configured.

4.5 Variables

The **yangcli-pro** program supports variables for easier reuse and script-based operations.

There are 2 types of variables:

- **file variables**: the variable name is a file name, and the contents of the variable are stored in this file.
- **internal variables**: the variable name is just an internal identifier, and the contents of the variable are stored in memory

Variables are set with assignment statements. Here are some examples:

```
guest@localhost> $$backup = get-config source=running
guest@localhost> $$bad-data = "warn"
guest@localhost> $itf = "//interface[name='eth0']"
```

Note that in order to assign a string value (e.g., $$bad-data = "warn" above), single or double quotes must be used. An unquoted string will be interpreted as a command name, not a simple string value.

Variables are referenced in a similar manner, except the variable is on the right-hand side of the equation. These commands are equivalent in this example:

```
guest@localhost> @myfile.xml = xget select=$itf
guest@localhost> @myfile.xml = xget //interface[name='eth0']
```

Complex variable substitution is also supported:

```
guest@localhost> copy-config source=$$backup target=candidate
```

Note that **yangcli-pro** will attempt to figure out the structure of the parameter (e.g., 'source' and 'target' above), and adjust the NETCONF operation content. In the example above, since 'source' and 'target' are choices, the real nodes within the cases are examined, and the most appropriate case is selected. The 'source' parameter will contain an in-line <config> element with all the child nodes in the
$$backup variable, and the target parameter will contain an empty element named **<candidate>.**

Complex variable substitution inside the strings is also supported:

```
guest@localhost> $x = 10
guest@localhost> $y = “interfaces”
guest@localhost> $z = “interface”
```

```
guest@localhost> $value = “ianaift:l2vlan”
guest@localhost> create /if:$y/if:$z[if:name='vlan$x']/if:type value=$value
```

There are several types of internal variables available in the **yangcli-pro** program:

- read-only system variables ($$USER)
- read-write system variables ($$default-operation)
- global user variables, available at all 'runstack' levels ($$backup)
- local user variables available in the current 'runstack' level only ($itf)

The command 'show vars' can be used to see the current value of all program variables:

```
yangcli-pro> sho vars

CLI Variables

yangcli-pro {
  aliases-file ~/.yumapro/.yangcli_pro_aliases
  alt-names true
  autoaliases true
  autocomp true
  autohistory true
  autoload true
  autonotif false
  autoconfig false
  autosessions true
  autotest true
  autouservars true
  bad-data check
  config-edit-mode level
  display-mode plain
  echo-notif-loglevel debug
  echo-notifs true
  echo-replies true
  feature-enable-default true
  fixorder true
  force-target candidate
  indent 2
  log-level info
  log-syslog-level info
  log-vendor-level info
  match-names one-nocase
  message-indent -1
  ncport 830
  private-key $HOME/.ssh/id_rsa
  prompt-type normal
  public-key $HOME/.ssh/id_rsa.pub
  script-input true
  subdirs true
  test-suite-file ~/.yumapro/yangcli_pro_tests.conf
  time-rpcs false
  time-rpcs-stats false
  time-rpcs-stats-file ~/yangcli_pro_rpc_stats.txt
  timeout 0
```

```
  transport ssh
  use-xmlheader true
  user andy
  uservars-file ~/.yumapro/yangcli_pro_uservars.xml
  warn-idlen 64
  warn-linelen 0
  wildcard-keys false
}

Read-only environment variables

  HOME /home/andy
  HOSTNAME
  LANG en_US.UTF-8
  PWD /home/andy
  SHELL /bin/bash
  USER andy
  YUMAPRO_DATAPATH
  YUMAPRO_HOME /home/andy/swdev/ypwork/netconf
  YUMAPRO_MODPATH
  YUMAPRO_RUNPATH

Read-write system variables

  aliases-file ~/.yumapro/.yangcli_pro_aliases
  alt-names true
  autoaliases true
  autocomp true
  autoconfig false
  autohistory true
  autoload true
  autonotif false
  autosessions true
  autotest true
  autouservars true
  bad-data check
  config-edit-mode level
  default-module
  default-operation merge
  display-mode plain
  echo-notif-loglevel debug
  echo-notifs true
  echo-replies true
  error-option none
  fixorder true
  indent 2
  log-level info
  match-names one-nocase
  message-indent -1
  optional false
  prompt-type normal
  script-input true
  server
  test-option set
  test-suite-file ~/.yumapro/yangcli_pro_tests.conf
  time-rpcs false
  time-rpcs-stats false
  time-rpcs-stats-file ~/yangcli_pro_rpc_stats.txt
  timeout 0
  use-xmlheader true
  user andy
  uservars-file ~/.yumapro/yangcli_pro_uservars.xml
  with-defaults none
```

```
No global variables

No local variables

yangcli-pro>
```

4.6 Scripts

Scripts are simply a collection of **yangcli-pro** commands and/or assignment statements that are stored in a text file, instead of typed directly. Scripts can call other scripts (except loops are not allowed), and numbered parameters are available (e.g., --P1='fred' passed as parameter, the $1 expands to 'fred' inside the script).

The **$YUMAPRO_RUNPATH** environment variable, or the **--runpath** configuration variable, can be used to set the directory path to look for script files. There is also a default path for finding files, explained in the YumaPro Tools User Manual.

The command '**list scripts**' can be used to show the potential script file available in the run path.

The command '**run foo**' is used to invoke a script named 'foo' (with no file extension).

If a command fails during a script, execution is halted right away and no more commands in the script are executed. If 'get-locks' was used, then any locks obtained will be automatically released. All script runstack levels will be canceled, not just the current script.

Script syntax will be expanded in a future release to provide loops and conditional statements.

5 toaster.yang

The toaster.yang module is included here for reference.

```
module toaster {

    namespace "http://netconfcentral.com/ns/toaster";

    prefix "toast";

    organization
        "Netconf Central, Inc.";

    contact
        "Andy Bierman <andy@netconfcentral.org>";

    description
        "YANG version of the TOASTER-MIB.";


    revision 2009-11-20 {
        description "Toaster module in progress.";
    }

    identity toast-type {
        description
          "Base for all bread types supported by the toaster.
           New bread types not listed here nay be added in the
           future.";
    }

    identity white-bread {
        description
          "White bread.";
        base toast:toast-type;
    }

    identity wheat-bread {
        description
          "Wheat bread.";
        base toast-type;
    }

    identity wonder-bread {
        description
          "Wonder bread.";
        base toast-type;
    }

    identity frozen-waffle {
        description
          "Frozen waffle.";
        base toast-type;
    }

    identity frozen-bagel {
        description
          "Frozen bagel.";
        base toast-type;
```

```
}

identity hash-brown {
    description
      "Hash browned potatos.";
    base toast-type;
}

typedef DisplayString {
    description
      "YANG version of the SMIv2 DisplayString TEXTUAL-CONVENTION.";
    reference "RFC 2579, section 2.";
    type string {
        length "0 .. 255";
    }
}

container toaster {
    presence
      "Indicates the toaster service is available";

    description
      "Top-level container for all toaster database objects.";

    leaf toasterManufacturer {
        type DisplayString;
        config false;
        mandatory true;
        description
          "The name of the toaster's manufacturer. For instance,
           Microsoft Toaster.";
    }

    leaf toasterModelNumber {
        type DisplayString;
        config false;
        mandatory true;
        description
          "The name of the toaster's model. For instance,
           Radiant Automatic.";
    }

    leaf toasterStatus {
        type enumeration {
            enum up {
              value 1;
              description
                "The toaster knob position is up.
                  No toast is being made now.";
            }
            enum down {
              value 2;
              description
                "The toaster knob position is down.
                  Toast is being made now.";

            }
        }
        config false;
        mandatory true;
        description
          "This variable indicates the current state of
           the toaster.";
```

```
    }
  }

  rpc make-toast {
      description
        "Make some toast.
         The toastDone notification will be sent when
         the toast is finished.
         An 'in-use' error will be returned if toast
         is already being made.
         A 'resource-denied' error will be returned
         if the toaster service is disabled.";
      input {
          leaf toasterDoneness {
              type uint32 {
                  range "1 .. 10";
              }
              default 5;
              description
                "This variable controls how well-done is the
                 ensuing toast. It should be on a scale of 1 to 10.
                 Toast made at 10 generally is considered unfit
                 for human consumption; toast made at 1 is warmed
                 lightly.";
          }
          leaf toasterToastType {
              type identityref {
                  base toast:toast-type;
              }
              default toast:wheat-bread;
              description
                "This variable informs the toaster of the type of
                 material that is being toasted. The toaster
                 uses this information, combined with
                 toasterDoneness, to compute for how
                 long the material must be toasted to achieve
                 the required doneness.";
          }
      }
  }

  rpc cancel-toast {
      description
        "Stop making toast, if any is being made.
         A 'resource-denied' error will be returned
         if the toaster service is disabled.";
  }

  notification toastDone {
      description
        "Indicates that the toast in progress has completed.";

      leaf toastStatus {
         description
           "Indicates the final toast status";
         type enumeration {
             enum done {
                description
                  "The toast is done.";
             }
             enum cancelled {
                description
                  "The toast was cancelled.";
```

```
               }
               enum error {
                  description
                    "The toaster service was disabled or
                     the toaster is broken.";
               }
            }
         }
    }


/*****************************************************

   Original TOASTER-MIB

TOASTER-MIB DEFINITIONS ::= BEGIN

IMPORTS
        enterprises
                FROM RFC1155-SMI
        OBJECT-TYPE
                FROM RFC-1212
        DisplayString
                FROM RFC-1213;

epilogue        OBJECT IDENTIFIER ::= {enterprises 12}
toaster         OBJECT IDENTIFIER ::= {epilogue 2}


toasterManufacturer OBJECT-TYPE
  SYNTAX  DisplayString
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
          "The name of the toaster's manufacturer. For  instance,
          Microsoft Toaster."
  ::= {toaster 1}

toasterModelNumber OBJECT-TYPE
  SYNTAX  DisplayString
  ACCESS  read-only
  STATUS  mandatory
  DESCRIPTION
          "The name of the toaster's model. For instance,
          Radiant Automatic."
  ::= {toaster 2}

toasterControl OBJECT-TYPE
  SYNTAX  INTEGER  {up (1), down (2)}
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
          "This variable controls the current state of the toaster.
           To begin toasting, set it to down (2). To abort toasting
          (perhaps in the event of an emergency), set it to up (2)."
  ::= {toaster 3}

toasterDoneness OBJECT-TYPE
  SYNTAX  INTEGER (1..10)
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
          "This variable controls how well-done is the
```

```
           ensuing toast. It should be on a scale of 1 to 10.
           Toast made at 10 generally is considered unfit
           for human consumption; toast made at 1 is warmed
           lightly."
  ::= {toaster 4}

toasterToastType OBJECT-TYPE
  SYNTAX  INTEGER {
                    white-bread (1),
                    wheat-bread (2),
                    wonder-bread (3),
                    frozen-waffle (4),
                    frozen-bagel (5),
                    hash-brown (6),
                    other (7)
                  }
  ACCESS  read-write
  STATUS  mandatory
  DESCRIPTION
          "This variable informs the toaster of the type of
           material that is being toasted. The toaster
           uses this information, combined with
           toasterToastDoneness, to compute for how
           long the material must be toasted to achieve
           the required doneness."
  ::= {toaster 5}

END
*************************************************************/
}
```