YangTools® User Manual

Table Of Contents

YangTools (0.9.6)

1.1 Legal Statements

1.2 Restricted Rights Legend

1.3 Additional Resources

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

YangTools® Installation and Quickstart Guide

Depending on the version of YangTools you purchased, other documentation includes:

YangTools® Manager Developer's Guide
YangTools® Agent Developer's Guide
YANG User Handbook

To obtain additional support you may email Netconf Central at the e-mail address support@netconfcentral.com

2 Preface

2.1 What is YangTools?

YangTools is a set of programs providing a complete network management system and development environment, which implements the following standards:

- Network Configuration Protocol (RFC 4741)
- NETCONF over SSH (RFC 4742)
- NETCONF Notifications (RFC 5277)
- SSH2 (RFC 4252 - 4254)
- XML 1.0
- XPath 1.0
- YANG Data modeling language (RFC xxxx)

The following programs are included in the YangTools suite:

- **yangdump**: validates YANG modules and uses them to generate other formats, such as HTML, XSD, SQL, and C source code
- **yangdiff**: reports semantic differences between two revisions of a YANG module, and generates YANG revision statements
- **yangcli**: NETCONF over SSH client, providing a simple but powerful command line interface for management of any NETCONF content defined in YANG
- **netconfd**: NETCONF over SSH server, providing complete and automated support for the YANG content accessible with the NETCONF protocol
- **netconf-subsystem**: thin client used to allow OpenSSH to communicate with the netconfd program. This is documented as part of the **netconfd** program, since they must be used together.
- **netconf-central**: Intranet version of the www.netconfcentral.org NETCONF/YANG documentation WEB server

Although any arbitrary YANG file can be automatically supported by YangTools, the following content (YANG modules) is built into the **netconfd** agent, and supported by the **yangcli** manager:

- **netconf.yang**: all the NETCONF protocol operations, including all YANG extensions to the NETCONF protocol (RFC 4741)
- **ietf-netconf-state.yang**: the standard NETCONF monitoring module in progress by the NETCONF WG (draft-ietf-netconf-monitoring-05.txt)

- **notifications.yang**: the standard NETCONF create-subscription command to start receiving NETCONF notifications (RFC 5277)
- **nc-notifications.yan**g: the standard NETCONF notifications (RFC 5277)
- **system.yang**: Proprietary system group and common notifications
- **nacm.yang**: Proprietary NETCONF Access Control Model
- **tests/*.yang**: Several modules are included for testing YANG and NETCONF behavior.

2.2 Intended Audience

This document is intended for users of the programs in the YangTools suite.
It contains the following information:

- Introduction to YANG and NETCONF based Network Management
- YangTools Configuration
- YangTools User Guides
- YangTools CLI Reference

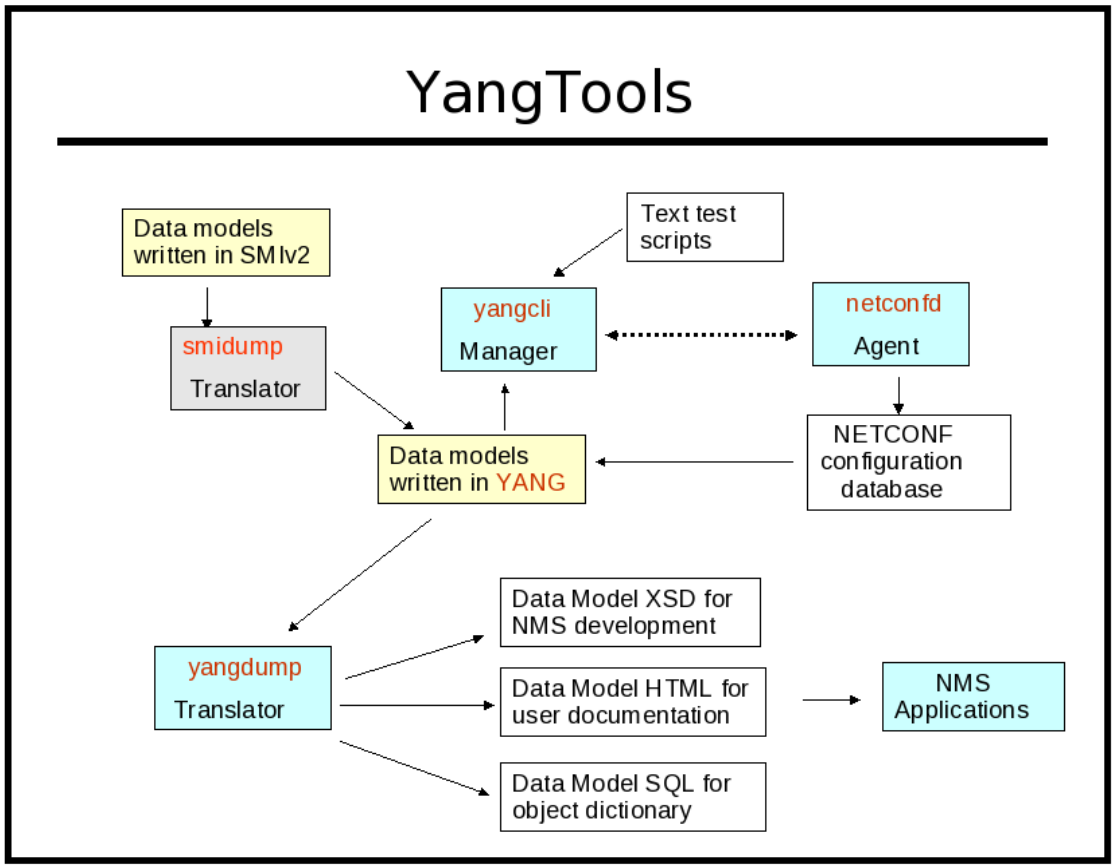
3 Introduction

The YangTools suite provides automated support for development and usage of network management information.

All management data is defined with the YANG data modeling language.

All management operations are encoded in XML 1.0 and performed with standard NETCONF protocol operations.

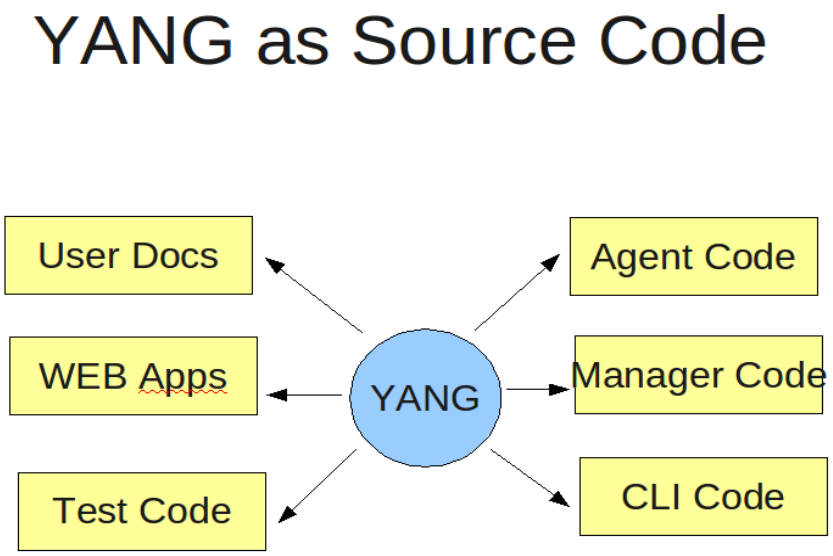
3.1 System Components



3.1.1 YANG

A YANG module define the semantics and syntax of a specific management feature. They are similar to SMIv2 (MIB) modules, but much more powerful and extensible. YANG provides the ability to define a detailed programmatic interface utilizing all protocol features:

- reusable derived data types
- reusable groupings of objects
- RPC operations
- database objects
- notifications



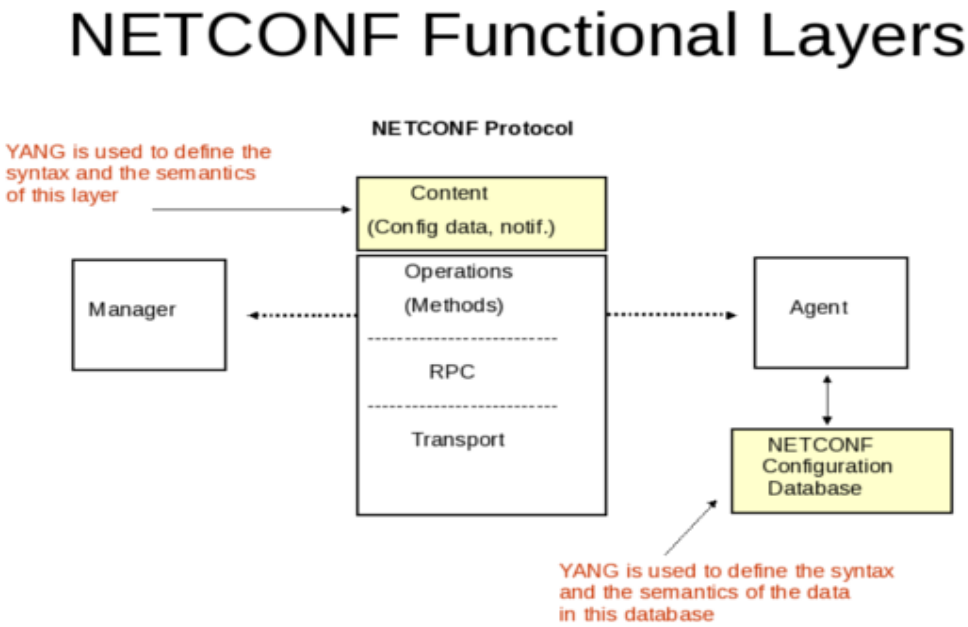
Network management software developers creating a new feature start by defining the YANG module(s) for the NETCONF representation of the feature. This can include any mixture of new operations, data, and notifications. Existing YANG modules can be augmented as well.

YANG provides complex nested data structures and choices, which allows data modelers to design management interfaces which closely resemble the native data structures within the agent implementation.

It is easy to get started with YANG, and there are many optional advanced features that can be utilized as well. YANG provides many machine-readable constructs which allow YangTools to automate many aspects of network management software development.

Semantics and details that are usually only found in 'description' clauses can be understood and implemented automatically by the software tools.

3.1.2 NETCONF



The NETCONF protocol is used to provide secure access all YANG content. The agent maintains a database which is accessed as if it was an XML instance document.

Data can be retrieved with XML (subtree) or XPath filters. Changes can be validated before being activated. Databases can be locked to prevent multiple managers from interfering with each other. Custom operations can be used to perform complex actions and perhaps return some data as well.

NETCONF can utilize several secure transport protocols. The mandatory transport (SSH2) is used by YangTools. The **OpenSSH** server is used in the **netconfd** implementation, and **libssh2** library is used in the **yangcli** implementation, to provide all SSH2 layer support.

By default, TCP port 830 (netconf-over-ssh) is used for all NETCONF communications between **yangcli** and **netconfd**. TCP port 22 (ssh) is also supported by default, and additional TCP ports can be configured.

NETCONF security is session-based. Privileges are granted to a session based on the username provided in the SSH connection setup.

Access control is configurable (via **nacm.yang**), based on group membership. The access control rules permit or deny access to one or more groups, to a subset of the YANG content. Separate defaults for read, write, and exec (RPC operation) access are provided.

3.1.3 YANG-BASED AUTOMATION

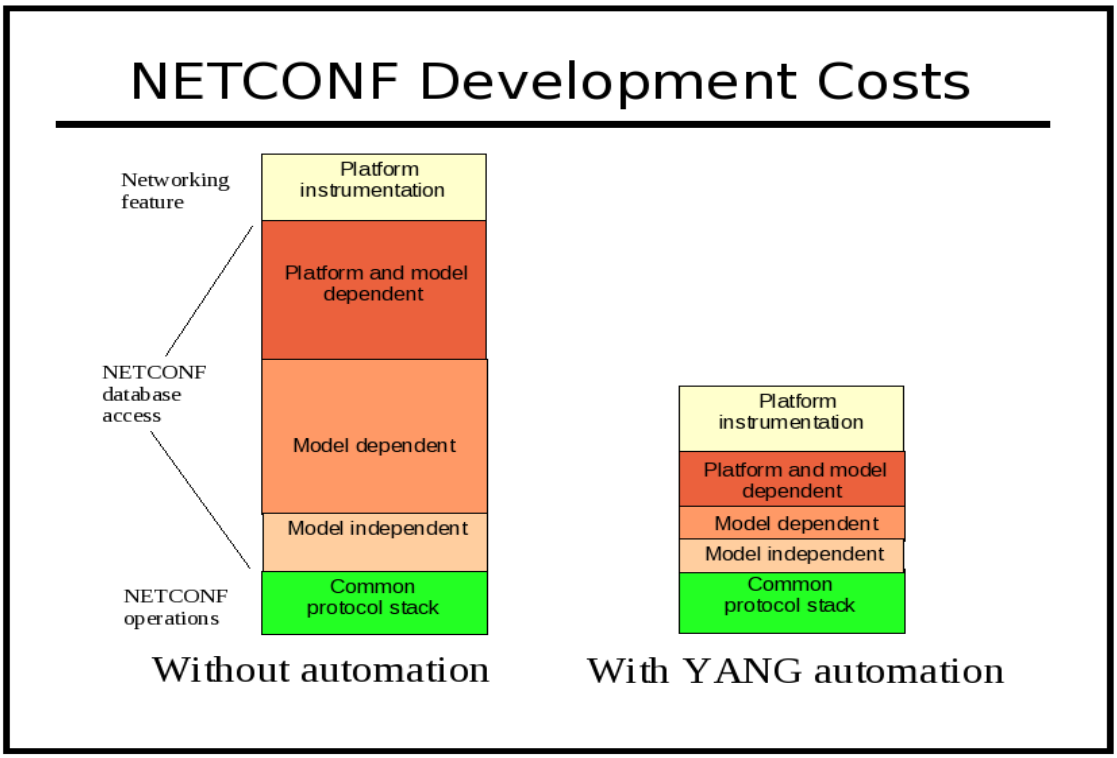
YangTools is a 100% “native YANG” implementation. This means that YANG modules are used directly by all the tools to control all aspects of NETCONF protocol usage. There are no lossy translations, or complicated configuration steps, in order to use a YANG module. Simply load a module and start using it.

The automation concepts will be familiar to SNMP developers who use SMIv2 to write MIB modules. The SMIv2 language contains enough machine-readable clauses so a manager and agent can automate certain aspects of the SNMP protocol implmentation.

YANG does the same thing for NETCONF developers, only 10 times better.

There are many more machine-readable constructs in YANG, and more powerful data modeling features. The complicated YANG features are optional, so traditional 'DESCRIPTION clause' based semantics are still supported.

The more machine-readable YANG clauses that are used, the more the **yangcli** manager and **netconfd** agent can automate the entire NETCONF protocol implementation.



The YANG language includes many ways to specify conditions for database validity, which traditionally are only documented in DESCRIPTION clauses:

YANG Automation Constructs

YANG statement	description
if-feature *feature-name;*	Construct containing the **if-feature** statement is only included if the specified feature is supported by the agent. Otherwise, the object does not exist on the agent.
revision *revision-date* { ... }	The **revision** statement identifies the most current version of a YANG module or sub-module. Multiple versions at once are supported in YANG.
import (by revision)	The import statement allows definitions from other modules to be used. A specific revision date can be used within the entire module. However, it is possible that different versions of imported typedefs and groupings can be used, if one imported module also imports some modules.
include (by revision)	The **include** statement provides the exact same features as the **import** statement, except it applied to sub-modules included within a module (or other sub-modules), instead of other modules. It allows multiple sub-modules to be combined to create one conceptual YANG module.
must *xpath-expr;*	If the object containing the **must** statement exists, then the XPath expression must evaluate to 'true' for the database to be valid. This provides referential integrity checks among related parameters.
when *xpath-expr;*	The object containing the **when** statement is only allowed to exist if the XPath expression evaluates to 'true'. This provides a SPARSE AUGMENTS capability when combined with the augment statement.
uses *grouping-name*;	The **uses** statement inserts an instance of a reusable **grouping**, replacing the uses node within the conceptual data tree.
refine *refine-target-path* { ... }	The **refine** statement is defined within a **uses** statement, and allows the specific grouping to be customized for each individual copy of the grouping contents.

	The tools can automatically support the refined objects, based on the sub-statements within the **refine** statement.
default *string*;	The **default** statement specifies the mandatory-to-use default value, if no leaf is provided. Unlike SMIv2 DEFVAL, it is not a suggestion, and the manager can rely on it. Defaults can be specified in **typedef** or **leaf** statements. If both are defined, then the leaf default will be used.
mandatory *boolean*;	The **mandatory** statement indicates that the choice, list or leaf must be provided by the manager. It will not be created by the agent. Most parameters are not mandatory however, so the default is 'false' if this statement is missing.
config *boolean*;	The **config** statement indicates if the object is writable, or read-only. The agent uses this information when automatically skipping config=false entries for the <get-config> operation.
key *key-leaf-list*;	The **key** statement indicates a set of one or more top-level leafs within the list that are used to name a specific instance of the particular list object. All protocol operations, such as <edit-config>, can be fully automated, based on the information in this statement.
unique *unique-node-lis*t;	The unique statement indicates an arbitrary tuple of descendant nodes within a list, which have to be unique within the list. These nodes are not keys, and can be nested anywhere within a single list entry.
min-elements *number*;	Specifies the minimum number of instances that a **list** or **leaf-list** object must have in a valid database. The default is zero, if this statement is not present.
max-elements *number* | 'unbounded' ;	Specifies the maximum number of instances that a **list** or **leaf-list** object can have in a valid database. The default is 'unbounded', if this statement is not present.
range *range-spec-string* ;	The **type** statement can specify the range of a numeric type. Since typedefs can be nested in YANG, the range statements are nested also, and constitute an AND expression (i.e., all the range tests must pass in the chain of type definitions.) THe

	keywords 'min' and 'max' indicate the minimum and maximum values from the parent typedef (if any), not the built-in type.
length *length-spec-string*;	The **length** statement is exactly like the **range** statement, except it limits the length of string **leaf** and **leaf-list** objects.
pattern *pattern-string* ;	The **pattern** statement specifies a regular expression that must evaluate to 'true' in order for the corresponding string **leaf** or **leaf-list** object to be valid. Multiple patterns encountered in a nested typedef chain must all evaluate to 'true' for the object to be valid.
deviation *deviation-target-path* { ... }	The **deviation** statement allows any YANG object be customized for a particular platform or implementation.. The tools can automatically support the altered objects, based on the sub-statements within the **deviation** statement. These changes can be of any nature, even those normally not allowed in YANG. The intent of the deviation statement os to accurately describe the object implementation, so the tools can automate the protocol operations correctly, even for non-standard implementations.
extension	The extension statement allows a vendor to add language extensions, and all YANG implementations must be able to parse the extension correctly. However, only implementations which actually understand the extension will support it. All others will simply ignore the extension.

3.1.4 YANG Language Extensions

There are several YANG extensions that are supported by YangTools. They are all defined in the YANG file named **ncx.yang**. They are used to 'tag' YANG definitions for some sort of automatic processing by YangTools programs. Extensions are position-sensitive, and if not used in the proper context, they will be ignored. A YANG extension statement must be defined (somewhere) for every extension used in a YANG file, or an error will be occur.

Most of these extensions apply to **netconfd** agent behavior, but not all of them. For example, the **ncx:hidden** extension will prevent **yangcli** from displaying help for an object containing this extension. Also, **yangdump** will skip this object in HTML output mode.

The following table describes the supported YANG language extensions. All other YANG extension statements will be ignored by YangTools, if encountered in a YANG file:

YANG Language Extensions

extension	description
ncx:hidden;	Declares that the object definition should be hidden from all automatic documentation generation. Help will not be available for the object in **yangcli**.
ncx:metadata "*attr-type attr-name*";	Defines a qualified XML attribute in the module namespace. Allowed within an RPC input parameter. **attr-type** is a valid type name with optional YANG prefix. **attr-name** is the name of the XML attribute.
ncx:no-duplicates;	Declares that the **ncx:xsdlist** data type is not allowed to contain duplicate values. The default is to allow duplicate token strings within an **ncx:xsdlist** value.
ncx:password;	Declares that a string data type is really a password, and will not be displayed or matched by any filter.
ncx:qname;	Declares that a string data type is really an XML qualified name. XML prefixes will be properly generated by **yangcli** and **netconfd**.
ncx:root;	Declares that the container parameter is really a NETCONF database root, like <config> in the <edit-config> operations. The child nodes of this container are not specified in the YANG file. Instead, they are allowed to contain any top-level object from any YANG file supported by the agent.
ncx:schema-instance;	Declares that a string data type is really an special schema instance identifier string. It is the same as an instance-identifier built-in type except the key leaf predicates are optional. For example, missing key values indicate wild cards that will match all values in **nacm** <dataRule> expressions.
ncx:secure;	Declares that the database object is a secure object. If the object is an **rpc** statement, then only the **netconfd** 'superuser' will be allowed to invoke this operation by default. Otherwise, only read access will be allowed to this object by default, Write access will only be allowed by the 'superuser', by default.

ncx:very-secure;	Declares that the database object is a very secure object. Only the 'superuser' will be allowed to access the object, by default.
ncx:xsdlist "*list-type*";	Declares that a string data type is really an XSD style list. **list-type** is a valid type name with optional YANG prefix. List processing within <edit-config> will be automatically handled by **netconfd**.
ncx:xpath;	Declares that a string data type is really an XPath expression. XML prefixes and all XPath processing will be done automatically by **yangcli** and **netconfd**.

3.1.5 YANG Compiler

The YangTools programs all use the same centralized YANG language parser.

The complete YANG language is supported, as defined in the latest version (draft-ietf-netmod-yang-06.txt). The file naming conventions defined in this specification must be used, along with all the language definition rules.

Definitions can be contained in modules and/or sub-modules.

Any number of revisions of a module or submodule can be used concurrently, The **import-by-revision** and **include-by-revision** features of YANG are fully supported, Refer to the section 'Searching for Files' for more details.

All extension usage within YANG files is supported and saved. The application data is available to all YangTools programs, including netconfd agent instrumentation. Refer to the 'YANG User Guide' for details on writing YANG files and using the extensions built into YangTools.

Note: The **smidump** is not part of YangTools, but it can be utilized to convert MIB modules written in SMIv2 into YANG modules, which can then be implemented in **netconfd**, and managed with **yangcli**. The freely available **libsmi** library contains the **smidump** program.

3.1.6 YANG Module Library

The central system component is the set of YANG data model modules which define all available management information. This set of modules is expected to grow over time, and there is usually a high degree of reuse and inter-dependence between the modules.

YANG modules can import other modules to reuse any definitions, and to augment objects in other modules. Each module represents one unique XML namespace used within the NETCONF protocol. A module can be partitioned into any number of submodules, each in a separate YANG file. The submodules are conceptually combined, and only the entire module is accessible to other modules.

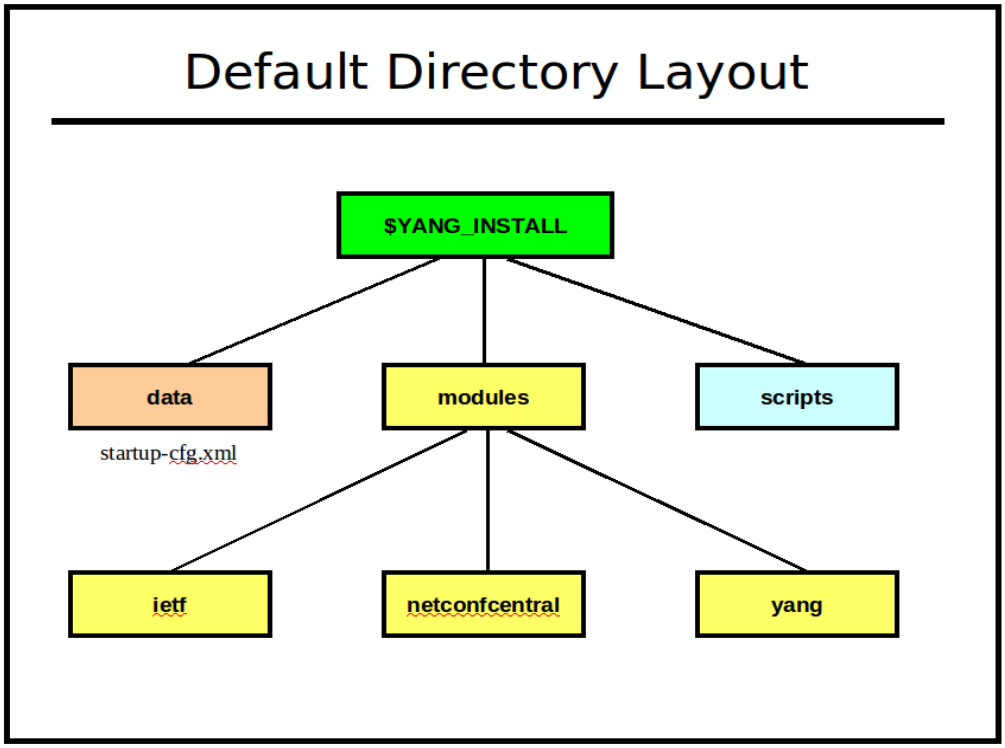
Directory Layout

YangTools can utilize several directories to store files used during operation. By default, a 'root' directory and all of its sub-directories are searched for these files. Several different roots can be searched. Generally, there is one centralized root (YANG_INSTALL) shared by all users, and one or more 'project' roots (YANG_HOME), which can be shared but may belong to a single user.

The YangTools programs need to find and store the following types of files during operations:

- YANG modules and submodules (*.yang):
- XML and text data files (usually *.txt or *.xml)
- command scripts for **yangcli**
- command-line-history file for **yangcli**

The search paths used to find these files are discussed in detail in the System Configuration section.

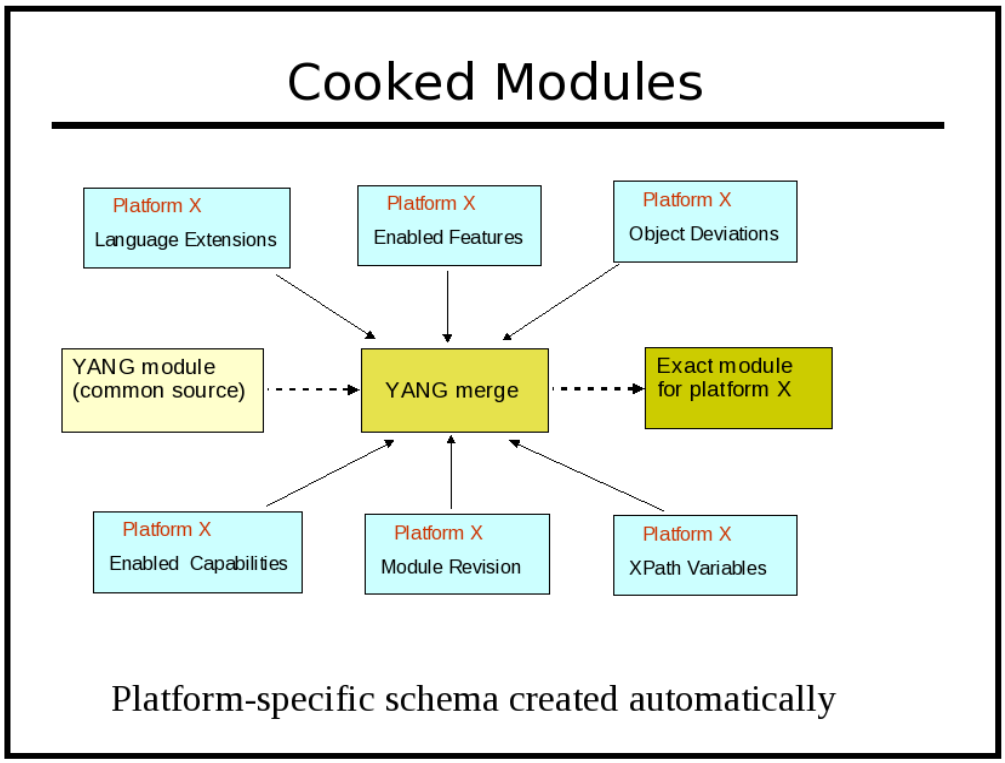


Module Revisions

YANG has extensive module lifecycle support. Each module or submodule has a revision date, and multiple revisions of the same module or submodule may be used at once within the same agent.

The YANG module repository is the authoritative source of common management information for the **netconfd** server. However, different platform implementations of the same data model need to be 'adjusted' slightly to reflect differences in the feature support available on

each platform. YangTools has an extensive set of mechanisms to automate the maintenance of these platform-specific 'special requirements'. A single YANG module (plus 'patches' and deviations as needed for each platform) can be published, instead of a separate version of the YANG module for each platform.



Module Naming Conventions

YANG module names are usually lower-case. Hyphen (-), underscore (_) and period (.) characters are allowed, after the first character, which must be a letter. It is suggested that only the hyphen (-) character be used as a separator between module name string components. YANG files must use the suffix '.yang'.

There are two forms of YANG file names: with and without a revision date.

module.yang

ietf-netconf-state.yang (no revision or unspecified revision)

module.revision-date.yang

ietf-netconf-state.2009-04-17.yang (must be the 2009-04-17 version)

These naming conventions are important when YangTools needs to resolve an 'import' or 'include' statement in a YANG file. Refer to section X.X for more details on YANG module search paths and the 'import-by-revision' feature of YANG.

3.1.7 YANG Files

YANG modules and submodules are text files encoded in UTF-8.

A module can be validated and checked for possible programming mistakes, by using the **yangdump** program. Many 'reports' can also be generated:

- exported symbols (--exports)
- imported modules (--dependencies)
- object identifiers (--identifiers)

The **yangdump** program is also used to generate other files, derived from the YANG content:

- **XML Schema Document (XSD)**: extends the NETCONF XSD with the YANG content layer definitions (--format=xsd)
- **HTML** or full file output: hyper-linked, color-coded formatting of YANG modules to support netconf-central or other WEB-based documentation system. There are several options for configuring the output, and all formatting is done with Cascading style-sheets (CSS) (--format=html)
- **netconf-central** documentation SQL database input file: supports the automated online documentation of YANG content (--format=sqldb). Refer to the netconfcentral.sql file for details on this output, in section X.X.
- **agent instrumentation codestubs**: the instrumentation callback functions, used in **netconfd** for activating specific YANG content, can be generated. This procedure is described in more detail in section X.X (--format=h, --format=c)
- **canonical YANG**: a YANG file can be reformatted so all statements are indented uniformly, and always appear in the same order. Objects maked as hidden (see the 'hidden' extension in ncx.yang) will not be generated. (--format=yang)
- **copy-YANG-and-set-name:** A YANG module can be validated and then copied (if no errors) to another location, adding the revision-date to the file name. (--format=copy)

3.1.8 NETCONF Manager

The NETCONF manager is an application that initiates and utilizes NETCONF sessions to control and monitor a NETCONF agent.

YangTools includes the **yangcli** application for this purpose. It can be used as a stand-alone tool with any NETCONF agent.

3.1.9 NETCONF Agent

The NETCONF agent is a server application that is always running on the managed device. It listens for NETCONF session requests from a NETCONF manager, and allows specific users to access specific subsets of the available content (operations, database access, and

notifications). It processes all incoming protocol operation requests from the manager, and insulates all the instrumentation code from these protocol operations.

YangTools includes the **netconfd** application for this purpose. It can be run on several different platforms, or easily adapted to embedded platforms.

3.2 Additional Resources

3.2.1 WEB Sites

3.2.2 Mailing Lists

3.2.3 FAQ

3.2.4 Reporting Problems

4 System Configuration

The YangTools programs use YANG to define its configuration parameters.

The 'ncx:cli' extension is used within a container with the same name as the program to define all CLI parameters. Some parameters are shared (see ncx-app-common.yang), so they are not located directly in the container.

```
container yangcli {
      ncx:cli;
      // yangcli CLI parameters defined as choices and leafs here
}
```

The following YANG modules are provided, which contain all the configuration parameters for YangTools:

- **ncx-types.yang**: contains common data types used in the YangTools applications
- **ncx-app-common.yang**: contains common CLI parameters used in all YangTools applications
- **ncx.yang:** contains YANG extensions used in any YANG module, including YangTools application modules
- **yangdump.yang:** configuration parameters for the **yangdump** application
- **yangdiff.yang**: configuration parameters for the **yangdiff** application
- **yangcli.yang**: configuration parameters and local commands for the **yangcli** application
- **netconfd.yang**: configuration parameters for the **netconfd** server

Note:

- The **netconf-subsystem** program does not have any configuration parameters at this time, so there is no YANG file defined for it.
- The **openssh** SSH server is configured separately, using the **sshd_config** file.
- The **libtecla** library, used by the yangcli program for command line editing support, has its own configuration file **~/.tecla**, to override the default (emacs) editing key assignments.

YangTools applications can accept configuration parameters from 3 sources, checked in the following order:

1. environment variables
2. command line parameters

3. configuration file

4.1 Environment Variables

The YangTools programs utilize system environment variables to customize and simplify configuration and operation of the programs.

These environment variables typically specify file search paths or default directory locations.

The following environment variables are used within YangTools:

- HOME
- YANG_HOME
- YANG_INSTALL
- YANG_MODPATH
- YANG_DATAPATH
- YANG_RUNPATH

4.1.1 $HOME

The **$HOME** environment variable contains the directory specification of the user's home directory, and is expected to be set by the system shell before use. The YangTools programs expect (by default) that sub-directories and files contained in this directory will be readable and writable.

Default value: none

CLI override: none

C shell example:

```
setenv $HOME  /home/andy
```

Bash shell example:

```
set $HOME=/home/andy
export HOME
```

4.1.2 $YANG_HOME

The **$YANG_HOME** environment variable contains the directory specification of the current YangTools project root directory.

Default value: none

CLI override: --yang-home

CLI example:

```
--yang-home=/home/andy/swdev/yangtools/trunk/netconf
```

C shell example:

```
setenv $YANG_HOME  /home/andy/swdev/yangtools/trunk/netconf
```

Bash shell example:

```
set $YANG_HOME=/home/andy/swdev/yangtools/trunk/netconf
export YANG_HOME
```

4.1.3 $YANG_INSTALL

The **$YANG_INSTALL** environment variable contains the directory specification of the YangTools installation root directory.

Default value: /usr/share/yang

CLI override: none

C shell example:

```
setenv $YANG_INSTALL  /usr/lib/yangtools
```

Bash shell example:

```
set $YANG_INSTALL=/usr/lib/yangtools
export YANG_INSTALL
```

4.1.4 $YANG_MODPATH

The **$YANG_MODPATH** environment variable contains a list of directory specifications that should be searched (in order) to find YANG modules and submodules. It can be used to extend the search path beyond the default locations.

The syntax for this parameter is a string containing the desired directory paths, separated by colon (:) characters. If the trailing forward slash (/) character is missing, then it will be added when searching for files.

By default, each entire directory and all its sub-directory contents will be searched for the requested file. This can be overridden with the 'no-subdirs' parameter. Refer to the Command Line Parameter Reference for more details. If *–no-subdirs* is used, then only the specified directory will be searched instead.

Note: This parameter specifies the exact directory locations when searching for files. This is different than the **$HOME**, **$YANG_HOME**, and **$YANG_INSTALL** environment variables, which specify a YangTools root directory.

Default value: none

CLI override: --modpath

CLI example:

```
--modpath="$HOME/modules2:/usr/local/modules"
```

C shell example:

```
setenv $YANG_MODPATH  "$HOME/modules2:/usr/local/modules"
```

Bash shell example:

```
set $YANG_MODPATH="$HOME/modules2:/usr/local/modules"
export YANG_MODPATH
```

4.1.5 $YANG_DATAPATH

The **$YANG_DATAPATH** environment variable contains a list of directory specifications that should be searched (in order) to find data files used by YangTools applications. It can be used to extend the search path beyond the default locations.

Data files used by the **yangcli** program are affected by this environment variable.

The location where the **netconfd** program keeps the file **startup-cfg.xml** is also affected by this environment variable. This file contains the contents of the non-volatile <startup> database, which is loaded into the <running> database when the agent boots.

The syntax for this parameter is a string containing the desired directory paths, separated by colon (:) characters. If the trailing forward slash (/) character is missing, then it will be added when searching for files.

By default, each entire directory and all its sub-directory contents will be searched for the requested file. This can be overridden with the 'no-subdirs' parameter. Refer to the Command Line Parameter Reference for more details. If *–no-subdirs* is used, then only the specified directory will be searched instead.

Note: This parameter specifies the exact directory locations when searching for files. This is different than the **$HOME**, **$YANG_HOME**, and **$YANG_INSTALL** environment variables, which specify a YangTools root directory.

Default value: none

CLI override: --datapath

CLI example:

```
--datapath=”$HOME/modules2:/usr/local/modules”
```

C shell example:

```
setenv $YANG_MODPATH  “$HOME/modules2:/usr/local/modules”
```

Bash shell example:

```
set $YANG_MODPATH=“$HOME/modules2:/usr/local/modules”
export YANG_MODPATH
```

4.1.6 $YANG_RUNPATH

The **$YANG_RUNPATH** environment variable contains a list of directory specifications that should be searched (in order) to find script files used by YangTools applications. It can be used to extend the search path beyond the default locations.

Script files used by the **yangcli** program are affected by this environment variable.

The syntax for this parameter is a string containing the desired directory paths, separated by colon (:) characters. If the trailing forward slash (/) character is missing, then it will be added when searching for files.

By default, each entire directory and all its sub-directory contents will be searched for the requested file. This can be overridden with the 'no-subdirs' parameter. Refer to the Command Line Parameter Reference for more details. If *–no-subdirs* is used, then only the specified directory will be searched instead.

Note: This parameter specifies the exact directory locations when searching for files. This is different than the **$HOME**, **$YANG_HOME**, and **$YANG_INSTALL** environment variables, which specify a YangTools root directory.

Default value: none

CLI override: --runpath

CLI example:

```
--runpath=”$HOME/scripts:/usr/local/scripts”
```

C shell example:

```
setenv $YANG_RUNPATH  “$HOME/scripts:/usr/local/scripts”
```

Bash shell example:

```
set $YANG_RUNPATH=“$HOME/scripts:/usr/local/scripts”
export YANG_RUNPATH
```

4.2 Searching for YANG Files

All YangTools programs search for YANG files in the same manner, using the same configuration parameters. The current working directory is included in this search path, so it is important to consider the directory in which a YangTools program is invoked. The search ends as soon as a suitable matching file is found.

There are two types of module searches:

1. searches on behalf of configuration parameters
2. searches on behalf of YANG import or include statements

The first term in a path specification may contain special character sequences:

- If the first character is the forward slash ('/'), then the entire path specification is used as an absolute path specification.

```
/usr/share/yang/modules
```

- If the first character is not the forward slash ('/'), and no special characters are found instead, then the entire path specification is used as an relative path specification, starting from the current working directory.

```
../more-modules/test7.yang
./this-dir/my-module.yang
testmodule.yang
old-modules/version7/
```

- If the first character is the tilde ('~') character, followed by the forward slash ('/') character, then the file search will start in the current user's $HOME directory .

```
~/modules/test/test.yang
```

- If the first character is the tilde ('~') character, followed by a user name, and then the forward slash ('/') character, then the file search will start in the specified user's $HOME directory . If the user is unknown, then the path specification is invalid.

```
~andy/modules/test/test.yang
~fred/scripts
```

- If the first character is the dollar sign ('$') character, followed by an environment variable name, and then the forward slash ('/') character, then the file search will start in the directory indicated by the contents of the environment variable. If the variable is unknown, or its contents do not represent a valid directory location, then the path specification is invalid.

```
$WORKDIR/tests/test-all-interfaces
$YANG_HOME/data/startup-cfg.xml
```

Note: Whenever YangTools searches a directory, it checks for the expected file type, but ignores the following:

- all files and sub-directories that begin with the period (.) character
- any directory named 'CVS'
- symbolic links for regular files

The following environment variables affect file searches:

- $HOME
- $YANG_HOME
- $YANG_MODPATH
- $YANG_DATAPATH
- $YANG_RUNPATH

The following configuration parameters affect file searches:

- --yang-home
- --modpath
- --datapath
- --runpath
- --no-subdirs

4.2.1 Parameter Searches

A parameter search is started on behalf of a CLI parameter, such as the **--module** parameter, used by the **yangdump** program. A search of this type can include directory path and file extension in the search parameter. If a filename with a file extension (must be '.yang') is given, then only that exact file will be checked. The current working directory will be used in this case, if no directory path (or a relative directory path) is provided.

```
--module=test.yang
--module=../more-modules/test3.2009-04-01.yang
```

If the exact filename is not found, then the search failed.
If a parameter based search does not have any directory path or file extension fields present, then a parameter search is the same as an import/include search.

4.2.2 Import/Include Searches

An import or include search is started on behalf of a YANG 'import' or 'include' statement. A search of this type includes only the module or submodule name, with no directory or file extension present. An optional 'revision-date' statement can be used in YANG, which means only a version of the YANG file with that exact current revision date will be used.

There are separate search algorithms, depending on whether the revision-date is used in the YANG import or include statement, and whether the imported or included module has a current revision statement.

Mode 1: import-by-revision

In this example, an import statement is causing a search for a module named 'foo' with a revision date of '2009-01-15'.
If a revision-date is used in the import or include statement, then the module search path will be checked as follows:

First, find a file with the same revision-date in the file name:

```
import foo {
    revision-date "2009-01-15";
    prefix foo;
}
```

If the file 'foo.2009-01-15.yang' is found, and the current revision statement in the module is equal to '2009-01-15', then the search is successfully terminated.

```
// file foo.2009-01-15.yang
module foo {

    namespace "http://example.com/ns/foo";
    prefix foo;

    // rest of header follows

    revision 2009-01-15 {
        description "Initial version.";
    }

    // rest of module follows
}
```

If the file is not found, or the most current revision date is not correct, then the module search is repeated for 'foo.yang'. If the file 'foo.yang' is found, and the current revision statement in the module is equal to '2009-01-15', then the search is successfully terminated.

```
// file foo.yang
module foo {

    namespace "http://example.com/ns/foo";
    prefix foo;

    // rest of header follows
```

```
    revision 2009-01-15 {
        description "Initial version.";
    }

    // rest of module follows
}
```

If the file is not found, or the most current revision date is not correct, then the module search failed.

Mode 2: import any revision

If no file name with the specified revision-date value is found, then the module search path is checked for a file with no revision-date in the file name:

```
import foo {
    prefix foo;
}
```

If the file 'foo.yang' is found, then it is used, regardless of the most current revision date (if any) found in the module. If it is not found then the module search failed.

Note: The first instance of 'foo.yang' in the module search path will be used, even if a more current version is available, later in the search path.

4.2.3 File Search Paths

YangTools uses configurable search paths to find the various files that are needed during operation.

Module Search Path

- If the module parameter is specified with a path or file suffix, the that filespec is tried, relative to the current working directory. If it is not found, or not the correct revision date, then the search terminates in failure.

```
--module=../test.yang
```

- If the module is specified without any path or file extension fields, then the module search path is checked, in order. The first step which produces a match terminates the

search successfully. If all steps are exhauted and no match is found then the search terminates in failure.

```
--module=foo
```

1. The current working directory is checked. No sub-directories are checked, if any are present.
2. Each directory specified in the **$YANG_MODPATH** environment variable, or set with the **–modpath** configuration parameter, is checked.
   - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
3. The **$HOME/modules** directory is checked.
   - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
4. The **$YANG_HOME/modules** directory is checked.
   - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
5. The **$YANG_INSTALL/modules** directory is checked.
   - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.

Data Search Path

YangTools programs can store data used during operation.

An example of a data file is the startup configuration file used by **netconfd**, usually called **startup-cfg.xml.**

1. If the file name has an absolute path specification, then that exact file location is tried. If no match is found, then the search will terminate in failure.
2. The current working directory is checked. No sub-directories are checked, if any are present.
3. Each directory specified in the **$YANG_DATAPATH** environment variable, or set with the **–datapath** configuration parameter, is checked.
   - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
2. The **$HOME/data** directory is checked.
   - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.

3. The **$YANG_HOME/data** directory is checked.
    - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.

Script Search Path

The **yangcli** program can store script files used during operation.

1. If the file name has an absolute path specification, then that exact file location is tried. If no match is found, then the search will terminate in failure.
2. The current working directory is checked. No sub-directories are checked, if any are present.
3. Each directory specified in the **$YANG_RUNPATH** environment variable, or set with the **–runpath** configuration parameter, is checked.
    - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
4. The **$HOME/scripts** directory is checked.
    - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
5. The **$YANG_HOME/scripts** directory is checked.
    - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
6. The **$YANG_INSTALL/scripts** directory is checked.
    - If the **--no-subdirs** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.

4.3 Configuration Files

The YangTools program configuration parameters can be stored in text or XML files.

The **--config** parameter is used to specify that configuration parameters should be retrieved from a file instead of the command line.

Any other configuration parameter (except **--config**) can be stored in a configuration file used for program input.

4.3.1 XML Configuration Files

The XML format for these files follows the structure of the NETCONF <config> element. Each parameter is stored within a container identifying the application which it is being configured. The **netconfd** stores its non-volatile <startup> database in this format. XML configuration file contents can appear in any order.

The following configuration parameters affect the generation and display of XML configuration files by **netconfd**:

- --indent
- --with-defaults

The following configuration parameter affects the location of XML configuration files by **netconfd**:

- --datapath
- $YANG_DATAPATH environment variable

Note : The IETF may standardize this container format soon. Do not rely on the top-level namespace URI. Any top-level element name <config>, in any namespace (even none), should be expected to contain a complete NETCONF database, or a subset of a NETCONF database.

The following example show some database objects from the NETCONF Access Control Model (nacm.yang), in XML configuration file format.

```
// file startup-cfg.xml
<?xml version="1.0" encoding="UTF-8"?>
<nd:config xmlns:nd="http://netconfcentral.com/ns/netconfd">
   <nacm:nacm xmlns:nacm="http://netconfcentral.com/ns/nacm">
      <nacm:groups>
         <nacm:group>
            <nacm:groupIdentity>nacm:admin</nacm:groupIdentity>
            <nacm:userName>andy</nacm:userName>
            <nacm:userName>fred</nacm:userName>
            <nacm:userName>barney</nacm:userName>
         </nacm:group>
      </nacm:groups>
      <nacm:rules>
         <nacm:moduleRule>
            <nacm:moduleName>netconf</nacm:moduleName>
            <nacm:allowedRights>read write exec</nacm:allowedRights>
            <nacm:allowedGroup>nacm:admin</nacm:allowedGroup>
         </nacm:moduleRule>
      </nacm:rules>
   </nacm:nacm>
</nd:config>
```

4.3.2 Text Configuration Files

The YangTools text configuration file format is based on some common Unix .conf file formats:

- A hash mark until EOLN is treated as a comment

```
# this is a comment
log-level info     # this is also a comment
```

- All text is case-sensitive
- Whitespace before or within a line is not significant
- The 'end a line' (EOLN) character ('\n') is used to end a command, so whitespace at the end of a line is significant.

- To enter a command on multiple lines, use an escaped EOLN (backslash-EOLN) for all but the last line

```
this is a command line
this is the start \
of a long \
three line command
this is a new command
```

- A YANG container parameter is represented by the container name, followed by a left curly brace ('{'), zero or more child nodes, and then a right curly brace ('}').

```
yangdump {
    # set some display control parameters
    log-level debug2
    warn-linelen 72
    indent 4
}
```

- A YANG list parameter is represented by the list name, followed by a whitespace separated sequence of key leaf values, followed by a left curly brace ('{'), zero or more child nodes, and then a right curly brace ('}').

```
ifStackEntry  11 42 {
      # the key leafs will also printed here
      ifStackHigherLayer 11
      ifStackLowerLayer 42
      ifStackStatus active
}
```

- Configuration files which are used with command line parameters may include program parameters for multiple applications.
  - Only the top-level container that matches the name of the program will be used.
  - Any other top-level containers will be ignored
  - Only the first instance of the desired program container will be used. Any additional containers will be ignored.

```
// test.conf
yangdump {
      # common yangdump parameters here
}

yangdiff {
      # common yangdiff parameters here
}
```

- Configuration file parameters can appear in any order. Only list index strings need to appear in their defined order.

The qualified names (identifiers, idenitityref, XPath) within a text configuration can be generated several ways. This is needed because sibling nodes in different XML namespaces can have the same name.

The –display-mode configuration parameter is used to control how textual display of values is done. It is an enumeration with four value:

display-mode values

enum value	description	example
plain	no prefix, just local-name	sysBootError
prefix	XML prefix and local-name	sys:sysBootError
module	module name and local name	system:sysBootError
xml	XML format	<sys:sysBootError>

When saving configuration files in non-volative storage, then it is suggested that only 'modprefix' or 'xml' mode be used, since these are the only deterministic modes. The set of XML prefixes in use at any one time is not persistent, and cannot be relied upon, unless the namespace declarations are saved as well (xml mode).

The following configuration parameters affect generation and display of text configuration files

- --indent
- --with-defaults
- --display-mode

4.4 Bootstrap CLI

Since YangTools programs use YANG to define CLI parameters, there needs to be an initial bootstrap CLI phase, in order to process parameters which affect the way YANG files are processed.

The bootstrap CLI is not as flexible as the main CLI processor, and the syntax is more strict. Parameters must be entered in either of the following forms:

- --name
- --name=value

If parameters are not entered in this format, then they will be skipped until the main CLI parameter processing is done. This may cause undesirable changes in key parameters, such as the module search path.

The following configuration parameters are also bootstrap parameters, and will take affect immediately, if entered from the command line:

- **--log**: log messages to the specified file instead of STDOUT
- **--log-level**: set the logging verbosity level
- **--log-append**: use the existing log file (if any) instead of overwriting it
- **--modpath**: use the specified module search path. This will override the **$YANG_MODPATH** environment variable, if it is set
- **--yang-home:** use the specified project root. This will override the **$YANG_HOME** environment variable, if it is set

Refer to the YangTools CLI Reference for more details. on these configuration parameters.

4.5 Configuration Parameters

Command line parameters are used to provide input to YangTools programs when they are invoked. They are also used extensively by the **yangcli** program, to represent RPC method input parameters and database nodes which are part of NETCONF operation content, such as the **<config>** parameter within the **<edit-config>** operation.

4.5.1 Parameter Syntax

A CLI parameter has 2 forms:

- Parameter contains a YANG type of 'empty' or a zero-length 'string':

  <prefix><parameter-name>

- Everything else:

  <prefix><parameter-name><separator><value>

There are up to 4 components in a CLI parameter:

1. **prefix**: consists of 0, 1, or 2 consecutive dash characters.
2. **parameter name**: name of the parameter. A partial name may be used if it is unique.
3. **separator**: either consists of the 'equals sign' character ('='), which may be preceded or followed by whitespace, or just whitespace with no equals sign character.
4. **value**: a quoted or unquoted string, an empty string is only allowed if quotes are entered.

The following example shows some ways the leaf 'foo' could be entered as a CLI parameter:

```
leaf foo {
  type uint32;
}

 foo=7
-foo=7
--foo=7
--foo =7
 foo 7
-foo 7
-foo = 7
--foo 7
--foo "7"
```

```
foo 7
```

4.5.2 NCX:CLI EXTENSION

The **ncx:cli** extension is used in in YANG container definitions, which represent the program CLI parameters, not NETCONF database parameters. It does not take any parameters, and is defined in **ncx.yang**.

```
container yangcli {
    ncx:cli;

    // all the yangcli CLI parameters
}
```

If this extension is present, then **netconfd** will ignore the container when loading the database object definitions. Only the program with the same name as the container will use the CLI parameter definition.

4.5.3 NCX:DEFAULT-PARM EXTENSION

The **ncx:default-parm** extension is used within a container with an **ncx:cli** extension, or within an 'input' section of an RPC operation definition. It is defined in **ncx.yang**.

If no parameter name is found when processing CLI parameter input, and the **ncx:default-parm** extension is present in the container or RPC input being processed, then the specified parameter name will be used instead of generating an error. The value must be valid for the parameter syntax, according to its YANG definition. This means that for the default parameter, only the <value> component of the complete parameter syntax may be used, as well as the normal forms.

```
container yangdump {
    ncx:cli;
    ncx:default-parm module;

    // all the yangdump CLI parameters
}
```

When invoking the **yangdump** program, the default CLI parameter is **--module**. These two command lines are equivalent:

```
yangdump --module=test1 --module=test2
```

```
yangdump test1 test2
```

A string that does not start with any dashes will still be tried as a parameter name, before trying the default parameter. If the value used for a default parameter conflicts with another parameter name, then the normal form must be used, instead of this form.

```
yangdump log-app  test1
```

Even if there was a module named 'log-app', it would not be tried as a **--module** parameter, since it also matches the **--log-append** parameter.

Note: the default parameter form is can be used in conjunction with shell wildcard characters, depending on the shell.

```
yangdump *.yang
```

```
yangdump --subtree=.
```

These commands are equivalent in the **yangdump** program.

5 yangdump User Guide

6 yangdiff User Guide

7 yangcli User Guide

7.1 Introduction

The **yangcli** program is a NETCONF over SSH manager application. It is driven directly by YANG modules, and provides a simple but powerful application interface for any YANG file. There is no configuration required at all to use any YANG file, although there are some YANG extensions that will be utilized if present.

7.1.1 Features

- Automatic support for all NETCONF protocol operations, including several 'short-hand' commands for the most common operations, like <edit-config> and <get-config>.
- Load any YANG module at boot-time or run-time and start using it immediately
- Supports NETCONF notifications, including :interleave capability
- Full XPath 1.0 and subtree filtering support
- Automatic support for all YANG language mechanisms, including extensions
- Any YANG **<rpc>** operation is automatically available as a **yangcli** command
- Uses YANG files directly as input, so no pre-processing or configuration needed to use a new module
- Can be called from **unix** scripts in 'batch-mode' to automatically establish a NETCONF session, issue a command or invoke a yangcli script, close the session, and return the results.
- Extensive interactive shell environment, including user variables, file variables, smart parameter set completion, and a simple scripting environment for automatic operation
- Automatic, context-sensitive (tab key) command-line completion
- Full support for XPath, instance-identifier, leafref, and identityref parameters
- Automatic, context-sensitive help system, based completely on YANG files and using the exact modules supported by the current NETCONF session, if connected
- Full, customizable command line editing, using **emacs** by default, but **vi** or a custom set of keystroke bindings are also supported
- Command line history and command recall
- Store and recall command line history files for later use
- Automatic NETCONF session management, including support for all YANG extensions to the <capability> element.
- Automatic recognition and support for all NETCONF 'capability' related operations.

- Automatic support for all YANG additions to the NETCONF protocol, such as the **insert** operation
- Unlimited nested scripts with up to 10 parameters each can automate testing and other management tasks

7.1.2 STARTING YANGCLI

The current working directory in use when **yangcli** is invoked is important. It is most convenient to run yangcli from a work directory, rather than the installation directory or within the module library.

The yangcli program can be invoked several ways:

- To get the current version and exit:

```
yangcli --version
```

- To get program help and exit:

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

- To start an interactive session with the default parameters:

```
yangcli
```

- To start an interactive session with a new log file:

```
yangcli --logfile=mylogfile
```

- To start an interactive session and append to an existing log file:

```
yangcli --logfile=mylogfile --log-append
```

- To get parameters from a configuration file:

```
yangcli --config=~/yangcli.conf
```

- To begin to connect to an agent upon startup, provide the **--agent** parameter. The **connect** command will be started upon startup and the user willl be prompted to enter the rest of the mandatory parameters to the **connect** command.

```
yangcli agent=myagent.example.com
```

- To connect to an agent and automatically connect without any interactive interruption, enter the **--agent**, **--user**, and **--password** parameters. A session startup will be attempted right away, using these parameters. Any optional parameters for the **connect** command (**--port** or **--timeout**) may be entered as well. All parameters can be entered from a config file, and/or the command line.

```
yangcli --agent=myagent.example.com \
      --user=andy --password=yangrocks
```

- To automatically connect to an agent, run a script in non-interactive mode, and then remain connected to the agent, add the **--run-script** parameter to the connection parameters. The **--runpath** parameter can also be entered, if needed.

```
yangcli --agent=myagent.example.com \
      --user=andy --password=yangrocks \
      --run-script=mytestscript
```

- To automatically connect to an agent, run a script in non-interactive mode, and then exit the program, add the **--batch-mode** and **--run-script** parameters to the connection parameters. The **--runpath** parameter can also be entered, if needed.

```
yangcli --agent=myagent.example.com \
      --user=andy --password=yangrocks \
      --run-script=mytestscript --batch-mode
```

- To automatically connect to an agent, and run a single command instead of a script, and then exit, use the **--run-command** parameter instead of the **--run-script** parameter. The **--batch-mode** parameter can be left out to remain in the current session (in interactive mode) after the command is invoked.

```
yangcli --agent=myagent.example.com \
      --user=andy --password=yangrocks \
      --batch-mode --run-command="sget /system"
```

7.1.3 STATEMENTS

The **yangcli** script interpreter accepts several types of statements:

yangcli Statements

type	description	example
command	invoke a local command and/or send an <rpc> to the agent	sget /system
variable assignment	set a user variable to some value	$system = sget /system
file assignment	set the contents of a file to some value	@save.txt = $system
variable deletion	delete a user variable or clear a system variable	$system =

Note: More statement types will be available in the next release.

7.1.4 COMMANDS

The yangcli program has several built-in commands, defined in y**angcli.yang**, **netconf.yang**. and **notifications.yang**.

The YANG **rpc** statement is used to define the syntax and behavior of each command.

There are 2 types of yangcli commands:

- **local**: the command is executed within the yangcli application, and can be invoked at any time.
- **remote**: the command is executed on the remote agent, and is only available when a NETCONF session is active. Any YANG **rpc** statement that **yangcli** does not recognize as a local command is treated as a remote command available on the agent.

Local Commands

command	description
cd	change the current working directory
connect	connect to an agent and start a NETCONF session
eventlog	view or clear the notification event log
fill	fill a user variable
help	get context-sensitive help
history	manage the command history buffer
list	list modules, objects, or other properties of the session
mgrload	load a YANG file into the manager only
pwd	print the current working directoty

quit	exit the program
recall	recall a line from the command history buffer
run	run a script
show	show variables and objects currently available

The following table shows the standard NETCONF protocol operations that are directly available for use, depending on the capabilities of the agent.

Standard NETCONF Commands

command	description
close-session	Close the current NETCONF session
commit	Make the candidate database be the running config
copy-config	Copy an entire NETCONF database
create-subscription	Start receiving NETCONF notifications
delete-config	Delete an entire NETCONF database
discard-changes	Discard any edits in the candidate database
edit-config	Alteration of the target database
get	Filtered retrieval of state data and running config
get-config	Filtered retrieval of any NETCONF database
get-schema	Get a data model definition file from the agent
kill-session	Force close another NETCONF session
lock	Lock a NETCONF database that is currently unlocked
unlock	Unlock a NETCONF database that is currently locked
validate	Validate the contents of a NETCONF database

The following **yangcli** commands are available for simplified access to standard NETCONF operations

Custom NETCONF Commands

command	description
create	Invoke an <edit-config> create operation
delete	Invoke an <edit-config> delete operation
insert	Invoke an <edit-config> YANG insert operation
merge	Invoke an <edit-config> merge operation
replace	Invoke an <edit-config> replace operation

save	Save the current edits on the agent in NV-storage
sget	Invoke a <get> operation with a subtree filter
sget-config	Invoke a <get-config> operation with a subtree filter
xget	Invoke a <get> operation with an XPath filter
xget-config	Invoke a <get-config> operation with an XPath filter

The following table shows the extended NETCONF protocol operations that are available on the **netconfd** agent only.

Extended netconfd Commands

command	description
load	Load a module into the agent
no-op	No operation
restart	Restart the agent
shutdown	Shutdown the agent

7.1.5 VARIABLES

The **yangcli** program utilizes several types of user-accessible variables. These variables can be listed with the '**show vars**' command.

A variable reference consists of 1 or 2 dollar signs ('$'), followed immediately by a valid identifier string (e.g., **$$global-log** or **$local-log**).

Variables can be 1 or more characters in length, and follow the YANG rules for identifier names. The first character must be a letter, 'A' to 'Z', or 'a' to 'z'. The 2nd to last characters can be a letter 'A' to 'Z', or 'a' to 'z', number ('0' to '9'), an underscore ('_'), a dash ('-'), or a period ('.') character.

There are 4 categories of parameters supported:

1. Read-only system variables
2. Read-write system variables
3. Read-write global user variables
4. Read-write local user variables

It is an error if a variable is referenced (in the right-hand-side of a statement) that does not exist.

The first 3 types are **global variables**, which means that they are available to all run-levels of all scripts. The last type, called a **local variable,** is only visible to the current run-level of the current script (or interactive shell). Refer to the following section for more details on run levels.

Variable Syntax

syntax	description	example
$$<variable-name>	Left hand side: set the global variable to some value	$$saved_get = get
$$<variable-name>	Right hand side: access the value of a global variable	fill --target=\ $$mytarget
$<variable-name>	Left hand side: set the local variable to some value	$myloglevel = \ $$log-level
$<variable-name>	Right hand side: access the value of any variable with this name (try local, then global)	$myuser = $user

The following table shows the **unix** environment variables that are available as read-only global variables in **yangcli**. These variables are set once when the program is started, and cannot be used in the the left hand side of an assignment statement.

Read-only system variables

variable	description
$$HOME	the **HOME** environment variable
$$HOSTNAME	the **HOST** or **HOSTNAME** environment variable
$$LANG	the **LANG** environment variable
$$PWD	the **PWD** environment variable, when **yangcli** was invoked
$$SHELL	the **SHELL** environment variable
$$USER	the **USER** environment variable
$$YANG_DATAPATH	the Y**ANG_DATAPATH** environment variable
$$YANG_HOME	the **YANG_DATAHOME** environment variable
$$YANG_MODPATH	the **YANG_MODPATH** environment variable
$$YANG_RUNPATH	the **YANG_RUNPATH** environment variable

The following table shows the CLI configuration parameters that can be read or changed (but not deleted). If a particular parameter was not set during program invocation, then the associated variable will contain the empty string.

Read-write system variables

variable	description
$$agent	the **--agent** configuration parameter
$$autocomp	the **--autocomp** configuration parameter
$$autoload	the **--autoload** configuration parameter
$$baddata	the **--baddata** configuration parameter
$$default-module	the **--default-module** configuration parameter
$$display-mode	the **--display-mode** configuration parameter
$$error-option	the **--error-option** configuration parameter
$$fixorder	the **--fixorder** configuration parameter
$$log-level	the **--log-level** configuration parameter
$$optional	the **--optional** configuration parameter
$$test-option	the **--test-option** configuration parameter
$$timeout	the **--timeout** configuration parameter
$$user	the **--user** configuration parameter
$$with-defaults	the **--with-defaults** configuration parameter

Read-write global user variables

If a unrecognized global variable (e.g., $$foo) is used in the left-hand side of an assignment statement, then a global user variable will be created with that name. If the global user variable already exists, then its value will be overwritten.

Read-write local user variables

If a local variable (e.g., $foo) is used in the left-hand side of an assignment statement, then a local user variable will be created with that name. If the local user variable already exists, then its value will be overwritten. If the variable is created within a script (i.e., run-level greater than zero), then it will be deleted when the script exits.

7.1.6 FILES

File contents can be used in **yangcli** statements, similar to user variables.

A file reference consist of the 'at-sign' character ('@') followed immediately by a valid file specification.

```
@foo.yang = get-schema --identifier=foo --version="" --format="YANG"

mgrload --module=foo
```

If the file extension is "**.yang**", "**.log**", "**.txt**", or "**.text**", then the value (or command output) will be saved, and yangcli will strip off the outermost XML (if needed) to save the requested file as a pure text file. Otherwise, the file will be saved in XML format.

Note: The **--display-mode** configuration parameter, and **$$display-mode** system variable, onlyt affect the output of data in file assignment statements if the file name has one of these extensions. Otherwise, the output will be in XML format.

Files may also be used as parameter replacements, within a command.

```
$saved_get = get --filter=@filter.xml --with-defaults=trim
```

It is an error if the file referenced does not exist or cannot be read.

7.1.7 Scripts

Any command can be entered at the interactive shell, or stored in a script file, and invoked with the '**run**' command. Scripts are simply text files, and the extension used does not matter.

There are no formal templates for scripts, like there are for RPC operations, at this time. Instead, positional parameters can be passed to any script.

The parameters named **--P1** to **--P9** allow up to 9 parameters to be passed to any script. Within each script, the numbered parameters '**$1**' to '**$9**' are available, and contain the value that was passed as the corresponding ---Pn" parameter when calling the script.

```
run connect --P1=andy --P2==localhost --P3=yangrocks

// connect script
# start a NETCONF session
$user = $1
$agent = $2
$password = $3

connect --user=$user --agent=$agent --password=$password
```

Run Levels

The **run** command can appear in a script.

When **yangcli** starts up, either in interactive mode or in batch mode, the script interpreter is at run level zero. Each time a **run** command is invoked, either at the command line or within a script currently being invoked, a new run level with the next higher value is assigned. Local variables are only visible within the current run level.

A maximum of 64 run levels are supported in **yangcli**.

7.1.8 Configuration Parameter List

The following configuration parameters are used by yangcli. Refer to the CLI Reference for more details.

yangcli CLI Parameters

parameter	description
--agent	Specifies the agent address to use in the **connect** command
--autocomp	Controls whether partial commands are allowed or not
--autoload	Controls whether modules used by the agent will be loaded automatically, as needed.
--bad-data	Controls how bad data about to be sent to the agent is handled.
--batch-mode	Indicates the interactive shell should not be used.
--config	Specifies the configuration file to use for parameters
--default-module	Specifies the default module to use to resolve identifiers
--display-mode	Specifies how values should be displayed
--fixorder	Controls whether PDUs are changed to canonical order before sending them to the agent.
--help	Get context-sensitive help
--indent	Specifies the indent count to use when writing data
--log	Specifies the log file to use instead of STDOUT
--log-append	Controls whether a log file will be reused or overwritten
--log-level	Controls the verbosity of logging messages
--modpath	Sets the module search path
--modules	Specifies a list of modules to load upon startup
--password	Specifies the password to use in the **connect** command
--port	Specifies the port number to use in the **connect** command

--run-command	Specifies the command to run at startup time
--run-script	Specifies the script to run at startup time
--timeout	Specifies the timeout to use in the **connect** command
--version	Prints the program version and exits
--warn-idlen	Controls how identifier lengths are checked
--warn-linelen	Controls how line lengths are checked
--warn-off	Suppresses the specified warning number
--yang-home	Specifies the **$YANG_HOME** project root to use when searching for files

7.2 Invoking Commands

Commands can be entered with parameters:

- in one continuous line

```
merge target=/toaster/toastControl --value="down"
```

- in several lines (using line continuation)

```
merge target=/toaster/toastControl \
   --value="down"
```

- interactively prompted for each parameter

```
merge
(will be prompted for target and value)
```

- a combination of the above

```
merge target=/toaster/toastControl
(will be prompted for value)
```

When a command is entered, and the yangcli script interpreter is running in interactive mode (**--batch-mode** not active), then the user will be prompted for any missing mandatory parameters.

If the **--optional** parameter is present (or the **$$optional** system variable is set to 'true'), then the user will be prompted for any optional parameters that are missing.

A command has the basic form:

<name (QName)> <parameter (any YANG type)>*

The command name is a qualified name matching the name used in the YANG rpc statement. This is followed by zero or more parameters (in any order).

A command parameter has the same syntax as a CLI configuration parameter.

The command name syntax is described below.

An un-escaped end-of-line character ('enter key') terminates command line input.

7.2.1 Command Prompt

The **yangcli** command prompt changes, depending on the context.

Idle Mode:

If the script interpreter is idle and there is no NETCONF session active, then the prompt is simply the program name:

```
yangcli>
```

If the script interpreter is idle and there is a NETCONF session active, then the prompt is the program name, followed by ' <user>@<agent>', depending on the parameters used in the **connect** command:

```
yangcli andy@myagent>
```

Continuation Mode:

If a backslash, end-of-line sequence ended the previous line, the prompt will simply be the word 'more' indented 3 spaces:

```
yangcli andy@myagent> get \
   more>
```

The 'more>' prompt will continue if the new line also ends in with an escaped end-of-line. When a new line is finally terminated, all the fragments are spliced together and delivered as one command line.

Note: context-sensitive command completion is not available in this mode.

Choice mode:

If a partial command has been entered in interactive mode, and the script interpreter needs to prompt for a YANG 'choice' parameter, then a list of numbered cases will be presented, and the prompt will be the same as it was before (depending on whether a NETCONF session is active or not), except a colon character (':'), followed by the command name, will be added at

the end. As long as parameters for the same command are being entered (i.e., prompted for child nodes within a selected case, the command name will be appended to the prompt.

```
yangcli andy@myagent> sget

Enter a number of the selected case statement:

  1: case varref:
       leaf varref
  2: case from-cli:
       leaf target
       leaf optional
       anyxml value

Enter choice number [1 - 2]:
yangcli andy@myagent:sget>
```

Parameter mode:

If a partial command has been entered in interactive mode, and the script interpreter needs to prompt for a leaf or leaf-list, then the parameter name will appear in angle brackets ('<' and '>').

```
Filling mandatory case /sget/input/from/from-cli:
Enter string value for leaf <target>
yangcli andy@myagent:sget>
```

If the '**ncx:password**' extension is part of the YANG definition for the leaf or leaf-list, then the characters entered at the prompt in this mode will not be echoed, and they will not be saved in the command history buffer. Any default value will not be printed either. Instead, 4 asterisks '****' will be printed, even though the correct value will be used in the command.

If a default value is available, it will appear in square brackets ('[' and ']'). In this case, entering 'return' will not be interpreted as an empty string, but rather the default value that was presented.

```
yangcli> connect
```

```
Enter string value for leaf <user> [andy]
yangcli:connect>

Enter string value for leaf <agent> [myagent]
yangcli:connect>

Enter string value for leaf <password> [****]
yangcli:connect>
Enter uint16 value for leaf <port> [830]

yangcli:connect>

Enter uint32 value for leaf <timeout> [30]

yangcli:connect>
```

Note: After a NETCONF session is terminated for any reason, the connection parameters will be remembered , and presented as defaults the next time the connect command is entered.

7.2.2 Command Name

The command name can be entered with or without an XML prefix:

```
yangcli andy@myagent> nc:get
yangcli andy@myagent> get
```

If there is a prefix (e.g., 'nc:get'), then it needs to be one of the XML prefixes in use at the time. Use the 'show modules' command to see the modules and prefixes in use. The YANG prefix will usually be the same as the XML prefix, but not always.

XML prefixes are required to be unique, so if any 2 YANG modules pick the same prefix, then 1 of them has to be changed for XML encoding purposes.

If the **--default-module** configuration parameter (or **$$default-module** system variable) is set, it will be used to resolve a command name without any prefix, if it is not a NETCONF or **yangcli** command.

An error message will be printed if the command entered cannot be found in any YANG module, or if there are multiple commands that match the same string.

7.2.3 ncx:default-parm Extension

Each command may define a default parameter, by placing an '**ncx:default-parm**' extension in the rpc input section in the YANG rpc statement. This extension allows less typing in **yangcli** to accomplish the same thing.

If the script interpreter encounters a string in the command line that is not a recognized parameter for that command, and there is a default parameter defined, then the string will be used as a value for the default parameter.

For example, the 'show' parameter is the default parameter for the 'history' command, so both of these commands will be interpreted to mean the same thing:

```
history --show=10
history 10
```

Note: the default parameter does not allow the wrong form of a parameter type to be entered, to accept the default for that parameter. For example, the 'show' parameter above has a default value of '25':

```
# this is the same as history show=25
history

# this is an error, not the same as the above
history show
```

The following table shows the default parameters that are available at this time.

Default Parameters

command	default parameter
cd	dir
connect	agent
create	target
eventlog	show
fill	target
help	command
history	show
insert	target
load	module
merge	target
mgrload	module
recall	index

replace	target
run	script
sget	target
sget-config	target
xget	select
xget-config	select

7.2.4 Parameter Mode Escape Commands

There are 4 escape sequence commands that can be used while entering parameters.

They all begin with the question mark character ('?'), and end with the 'Enter' key.

Control key sequences are not used because that would interfere with command line editing keys.

Parameter mode escape sequences

escape sequence	description
?	Print some help text
??	Get all available help text
?s	Skip this parameter
?c	Cancel this command

Note: If the current parameter is considered hidden (**ncx:hidden** extension used in the YANG definition), then no help will be available for the parameter, even though it is accessible. This also apples to the **help** command. Any command or parameter designated as **ncx:hidden** will be treated as an unknown identifier, and no help will be given.

Note: Skipping mandatory nodes with the '?s' command is affected by the **--bad-data** configuration parameter and **$$bad-data** system variable. An error, warning, or confirmation check may occur. Refer to the CLI Reference for more details.

Note: If there are any YANG defined values (e.g., enumeration, bits, default-stmt) available for the current parameter, then pressing the tab key will list the full or partial completions available.

7.2.5 Using XPath Expressions

There are some command parameters, such as the **--target** parameter for the **create** command, that accept XPath absolute path expressions.

If prefixes are present, then they must match the set of XML prefixes in use at the time. Use the **show modules** command to see the current set of prefixes.

If prefixes are not used, then the first available matching XML namespace will be used instead.

If the starting forward slash character ('/') is missing, then it will be added.

```
# these are all the same value
yangcli:fill> system
yangcli:fill> /system
yangcli:fill> /sys:system
```

It is important to remember 2 simple rules to avoid common errors in XPath expressions:

1. String constants must be quoted with single quote characters.
   The expression [name=fred] is not the same as [foo='fred'].
   The former compares the 'name' node value to the 'fred' node value.
   The latter compares the 'name' node value to the string 'fred'.
2. The double quote character ('"') is not allowed in XPath **--select** parameter expressions because the expression will be sent to the agent inside a double-quoted string.

If an incomplete XPath absolute path expression is entered, and the script interpreter is in interactive mode, then the user will be prompted to fill in any missing mandatory nodes or key leafs.

```
# complete form of ifMtu leaf
yangcli:fill> /interfaces/interface[name='eth0']/ifMtu

# incomplete form of ifMtu leaf
yangcli:fill> /interfaces/interface/ifMtu

Filling key leaf <name>:
Enter string value:
```

The **--select** parameter for the **xget** and **xget-config** commands accepts full XPath expressions. The expression must yield a node-set result in order to be used as a filter in NETCONF **get** and **get-config** operations.

One of the simplest XPath filters to use is the **descendant-or-self** filter ('//<expr>').

For example, this command will retrieve all instances of the 'ifMtu' leaf:

```
xget //ifMtu
```

When interface (or any list) entries are returned by netconfd, they will contain the the entire path back to the root of the YANG module, not just the specified node. Also, any key leafs within a list are added. This is only done if the XPath expression is missing any predicates for key leafs.

This is different than XPath 1.0 as used in XSLT. Since NETCONF **get** and **get-config** operations return complete XML instance documents, not node-sets, the ancestor nodes and

naming nodes need to be added.

```
# reply shown with --display-mode=plain

data {
   interfaces {
     interface eth0 {
               name eth0
               ifMtu 1500
   }
  interface eth1 {
                name eth1
                ifMtu 1518
   }
}
```

7.2.6 Special Parameter Handling

Some special handling of YANG data structures is done by the script interpreter.

Containers

Some parameters, such as the **--source** and **--target** parameters in many commands, are actually represented as a container with a single child -- a choice of several leaf nodes. In this situation, just the name of the desired leaf node can be entered (when in idle mode), as the 'contents' of the container parameter.

```
sget-config /system source=candidate
```

Choices and Cases

If a parameter name exact-match is not found, and a partial match is attempted, then choice and case node names will be ignored, and not cause a match.

Since these nodes never appear in the XML PDUs they are treated as transparent nodes (wrt/ parameter searches) unless they are specified with their full name.

Parameters that are a choice of several nodes, similar to above, except without a parent container node, (e.g., **--help-mode**) can be omitted. The accessible child nodes within the case nodes can be entered directly (e.g., **sget --target** parameter).

```
# this is not allowed because 'help-mode' is not complete
yangcli> help --command=help --help-mo=brief

# this is allowed because 'help-mode' is complete,
```

```
# even though help-mode is a choice and 'brief' is
# an empty leaf
yangcli> help help help-mode=brief

# choice and case names are transparent when
# searching for parameter names, so the
# following command is the same as above
yangcli> help help brief
```

Lists and Leaf-Lists

When filling a data structure and a descendant node is entered, which is a YANG list or leaf-list, then multiple entries can be entered. After the node is filled in, there will be a prompt (Y/N, default no) to add more list or leaf-list entries.

Binary Data Type

The YANG binary data type is supported. Parameters of this type should be entered in plain text and they will be converted to binary format.

7.2.7 Command Completion

The 'tab' key is used for context-sensitive command completion:

- If no command has been started, a list of available commands is printed
- If a partial command is entered, a list of commands which match the characters entered so far is printed
- If a command is entered, but no parameters, then a list of available parameters is printed
- If a command is entered, and the cursor is within a command name, then a list of parameters which match the characters entered so far is printed
- If a command is entered, and the cursor is after a command name, but not within a value string, then a list of available parameters is printed
- If a command is entered, and the cursor is within a command value, then a list of possible values which match the characters entered so far is printed. Note that not all data types support value completion at this time.
- If no values are available, but a default value is known, that value will be printed

Command list example: no NETCONF session is active:

```
yangcli> <hit tab key>
cd       fill     history  mgrload  quit     run
connect  help     list     pwd      recall   show
```

Command list example: NETCONF session is active

```
yangcli andy@myagent.example.com> <hit tab key>
cd                    get-schema            recall
close-session         help                  replace
commit                history               restart
connect               insert                run
copy-config           kill-session          save
create                list                  sget
create-subscription   load                  sget-config
delete                load-config           show
delete-config         lock                  shutdown
discard-changes       merge                 unlock
edit-config           mgrload               validate
fill                  no-op                 xget
get                   pwd                   xget-config
get-config            quit
```

7.2.8 Command Line Editing

The command line parser is based on **libtecla**, a freely available library.

The home page is located here:

http://www.astro.caltech.edu/~mcs/tecla/

The complete user guide for configuring **libtecla** is located here:

http://www.astro.caltech.edu/~mcs/tecla/tecla.html

If the file **$HOME/.teclarc** exists, then **libtecla** will use it to configure the key bindings.

By default, **libtecla** uses **emacs** key bindings. There is no need for any further **libtecla** configuration if **emacs** mode is desired.

In order to use the **vi** editor key bindings, the **$HOME/.teclarc** file must exist, and it must contain the following line:

```
edit-mode vi
```

Custom key bindings are also available. Refer to the **libtecla** documentation for more details on command line editing key customization.

The control key sequence (^F == control key and f key at the same time). The letter is not case-sensitive, so ^F and ^f are the same command.

The alt key sequence (M-f == alt key and f key at the same time). The letter is not case-sensitive, so M-F and M-f are the same command.

The following table shows the the most common default key bindings:

command	description
^F	cursor right
^B	cursor-left
^A	beginning of line
^E	end of line
^U	delete line
M-f	forward-word
M-b	backward word
^P	up history
^N	down history

7.2.9 Command History

Each command line is saved in the command history buffer, unless a password is being entered in parameter mode.

By default, the previous history line (if any) will be shown if the ^P key is pressed.

By default, the next history line (if any) will be shown if the ^N key is pressed.

In addition, the **history** command can be used to control the command line buffer further. This command has 4 sub-modes:

- **show**: show maximum of N history entries (default is 25)
- **clear**: clear the history buffer
- **save**: save the history buffer to a file
- **load**: load the history buffer from a file

Refer to the Command Reference section for more details on the **history** command.

7.2.10 Command Responses

The command output and debugging messages within yangcli is controlled by the current log level (error, warn, info, debug, debug2, debug3).

If a command is executed by the script interpreter, then a response will be printed, depending on the log level value.

If the log level is 'info' or higher, and there were no errors and no response, then the string 'OK' is printed.

```
yangcli> $foo = 7
   OK
yangcli>
```

If the log-level is set to 'error' or 'warn', then the 'OK' messages will be suppressed.

If the log level is set to 'debug' or higher, then responses from the agent will be echoed to the log (file or STDOUT). The current display mode will be used when printing data structures such as <rpc-error> and <notification> element contents.

If an error response is received from the agent, it will always be printed to the log.

```
yangcli andy@myagent> create /system

Filling container /system:
RPC Error Reply 5 for session 8:

rpc-reply {
   rpc-error {
      error-type application
      error-tag access-denied
      error-severity error
      error-app-tag limit-reached
      error-path /nc:rpc/nc:edit-config/nc:config/sys:system
      error-message 'max-access exceeded'
   }
}

yangcli andy@myagent>
```

Refer the the **--log-level** parameter in the CLI Reference for more details.

7.3 NETCONF Sessions

The **yangcli** program can be connected to one NETCONF agent at a time.

Run multiple instances of yangcli to control multiple agents at once.

Use the connect command to start a NETCONF session.

This section explains how to use **yangcli** to manage a NETCONF agent, once a session is established.

When a NETCONF session starts up, a <capability> exchange is done, and the agent reports exactly what content it supports. This information is used extensively to customize the session, and report errors and warnings for the session.

7.3.1 Connection Startup Screen

If the **--log-level** is set to 'info' of higher, then a startup screen will be displayed when a NETCONF session is started. It contains:

- startup banner
- manager session ID
- agent session ID
- protocol capabilities supported by the agent
  - Includes revision-date of supported module
- YANG modules supported by the agent
  - Includes any features and deviations supported in the module
- Enterprise specific capabilities supported by the agent
- Default target database (<candidate> or <running>)
- Save operation mapping for the agent
- **with-defaults** reporting capability reported by the agent

The following example shows a typical startup screen connecting to the **netconfd** agent:

```
NETCONF session established for andy on myagent

Manager Session Id: 1
Agent Session Id: 8

Agent Protocol Capabilities
   Protocol Version: RFC 4741
   candidate:1.0
   rollback-on-error:1.0
   validate:1.0
```

```
   xpath:1.0
   notification:1.0
   interleave:1.0
   with-defaults:1.0
   netconf-monitoring:1.0
   schema-retrieval:1.0

Agent Module Capabilities
   ietf-inet-types/2009-05-13
   ietf-netconf-state/2009-03-03
   ietf-with-defaults/2009-04-10
   ietf-yang-types/2009-05-13
   nacm/2009-05-13
   nc-notifications/2008-07-14
   ncx/2009-06-12
   ncx-app-common/2009-04-10
   ncxtypes/2008-07-20
   netconfd/2009-05-28
   notifications/2008-07-14
   system/2009-06-04
   test/2009-06-10
      Features:
         feature1
         feature3
         feature4

Agent Enterprise Capabilities
   None

Default target set to: <candidate>
Save operation mapped to: commit
Default with-defaults behavior: report-all
Additional with-defaults behavior: trim:explicit


Checking Agent Modules...
```

```
yangcli andy@svnserver>
```

7.3.2 Agent Tailored Context

While a NETCONF session is active, the set of available YANG modules will be set to the modules that the agent is using.

If the module contains YANG features that are not advertised in the <capabilities> exchange, then those data definitions will not be available (by default) for use in **yangcli** commands.

If the module contains an object with a 'when' statement, and the 'when' XPath expression evaluates to 'false', then that data definition will not be available (by default) for use in **yangcli** commands.

The **help** command will be tailored to the modules, capabilities, features, and module deviations reported by the agent in <capability> exchange.

7.3.3 Retrieving Data

There are 6 commands available to retrieve generic data (i.e., an arbitrary subset of an entire NETCONF database):

command	description
get	raw NETCONF <get> operation
get-config	raw NETCONF <get-config> operation
sget	high-level subtree filter, using the <get> protocol operation
sget-config	high-level subtree filter, using the <get-config> protocol operation
xget	high-level XPath filter, using the <get> protocol operation
xget-config	high-level XPath filter, using the <get-config> protocol operation

All the high-level retrieval operations support the **$$with-defaults** system variable. The <with-defaults> parameter will be added the the NETCONF PDU if this variable is set to a value other than 'none' (the default). This system variable will be used as the default if not entered directly.

```
sget /system --with-defaults=$$with-defaults
```

This parameter can also be specified directly, each time the command is used.

```
xget-config //ifMtu --with-defaults=trim
```

The **$$bad-data** system variable is used to control how invalid operations and data are sent to the agent. The **xget** and **xget-config** commands are affected by this parameter. If the **:xpath** capability was not advertised by the agent when the session started, an error or warning may occur if these commands are used.

If any data is received that **yangcli** does not understand, then a warning message will be printed and the data will be treated as if it was defined with the YANG '**anyxml**' data type.

7.3.4 Modifying Data

The following commands are available to modify generic data (i.e., an arbitrary subset of an entire NETCONF database):

command	description
commit	raw NETCONF <commit> operation
create	high-level <edit-config> operation, with nc:operation='create'
delete	high-level <edit-config> operation, with nc:operation='delete'
delete-config	raw NETCONF <delete-config> operation
discard-changes	raw NETCONF <discard-changes> operation
edit-config	raw NETCONF <edit-config> operation
fill	fill a variable for re-use in other operations
insert	high-level <edit-config> operation, with YANG insert operation extensions
lock	lock a NETCONF database
merge	high-level <edit-config> operation, with nc:operation='merge'
replace	high-level <edit-config> operation, with nc:operation='replace'
save	High level save operation, depending on the default target (candidate or running)
unlock	unlock a NETCONF database

All the high-level editing operations use the **--target** parameter reported by the agent when the session started up. If the agent did not report the **:candidate** or **:writable-running** capabilities, then there will be no writable target, and an error will occur if these commands are entered.

All the high-level editing operations support the **$$test-option** system variable. The <test-option> parameter will be added the the NETCONF <edit-config> PDU if this variable is set to

a value other than 'none' (the default). This system variable will be used as the default if not entered directly.

```
replace /interfaces/interface[name='eth0']/ifMtu \
      --test-option=$$test-option \
      --value=$newvalue
```

This parameter can also be specified directly, each time the command is used.

```
$newvalue = 1518

replace /interfaces/interface[name='eth0']/ifMtu \
      --test-option=test-only \
      --value=$newvalue
```

All the high-level retrieval operations support the **$$error-option** system variable. The <error-option> parameter will be added the the NETCONF <edit-config> PDU if this variable is set to a value other than 'none' (the default). This system variable will be used as the default if not entered directly.

```
replace /interfaces/interface[name='eth0']/ifMtu \\
      --error-option=$$error-option \
      --value=1518
```

This parameter can also be specified directly, each time the command is used.

```
replace /interfaces/interface[name='eth0']/ifMtu \
      --error-option=rollback-on-error \
      --value=1518
```

The high level **save** command is mapped to other commands, depending on the capabilities reported by the agent.

save command

capabilities	real command(s)
:candidate	commit
:writable-running	<none>
:startup	copy-config --source=running \ --target=startup

7.3.5 Using Notifications

The **create-subscription** command is used to start receiving notifications.

The netconfd agent will include a <sequence-id> element in any notification that is saved in the replay buffer. This unsigned integer can be used to help debug notification filters (i.e., if non-consecutive <sequence-id> values are received, then the notification was filtered, or dropped due to access control policy).

If any replay notifications are desired, then the **--startTime** parameter must be included. At the end of the stored notifications, the agent will send the <replayComplete> event. This notification type is not saved, and will not be found in the agent replay buffer, if replay is supported by the agent. The **netconfd** agent will not include a <sequence-id> element in this notification type.

If the notification subscription should stop at a certain time, then the **--stopTime** parameter must be included. At the end of the stored notifications, the agent will send the <replayComplete> event, followed by the <notificationComplete> event. . This notification type is not saved, and will not be found in the agent replay buffer, if replay is supported by the agent. The **netconfd** agent will not include a <sequence-id> element in this notification type.

Notifications are printed to the log, using the current **$$display-mode** system variable setting, when and if they are received.

Notifications are also logged. Use the **eventlog** command to access the notifications stored in the event log.

7.3.6 Configuration Parameters That Affect Sessions

The **--agent**, **--user**, and **--password** configuration parameters can be used to start a NETCONF session automatically at startup time. The connect command will only be attempted at startup time if the **--agent** parameter is present.

If all 3 of these parameters are present at startup time, then no interactive prompting for additional optional parameters will be done. Instead the connect command will be invoked right away.

During normal operation, the **--optional** configuration parameter, or the **$$optional** system variable, can be used to control interactive prompting for optional parameters.

The **--agent** parameter is saved in the **$$agent** system variable, which can be overwritten at any time. If set, this will be used as the initial default value for the **--agent** parameter in the **connect** command.

The **--fixorder** configuration parameter can be used to control XML PDU ordering. If set to 'true', then the PDU will be reordered (if needed),to use the canonical order, according to the YANG specification. If 'false', the order parameters are entered at the command line will be their NETCONF PDU order as well. The default is 'true'. To send the agent incorrectly ordered data structures on purposes, set this parameter to 'false'.

The **--user** parameter is saved in the **$$user** system variable, which can be overwritten at any time. If set, this will be used as the initial default value for the **--user** parameter in the **connect** command.

The **--with-defaults** configuration parameter, or the **$$with-defaults** system variable, can be used to set the default value for the 'with-defaults' parameter extension for the NETCONF **get**, **get-config**, and **copy-config** protocol operations. The default is 'none'.

The **--error-option** configuration parameter, or the **$$error-option** system parameter, can be used to set the default value for the **--error-option** parameter for the NETCONF edit-config protocol operation. The default is 'none'.

The **--test-option** configuration parameter, or the **$$test-option** system parameter, can be used to set the default value for the **--test-option** parameter for the NETCONF edit-config protocol operation. The default is 'none'.

The **--bad-data** configuration parameter, or the **$$bad-data** system variable, can be used to control how **yangcli** handles parameter values that are known to be invalid, or usage of optional protocol operations that the current session does not support. The default value is 'check'. To use **yangcli** in a testing mode to send the agent incorrect data on purpose, set this parameter to 'warn' or 'ignore'.

7.3.7 Trouble-shooting NETCONF Session Problems

If the NETCONF session does not start for any reason, one or more error messages will be printed, and the prompt will indicate 'idle' mode. This section assumes that the agent is netconfd, and these debugging steps may not apply to all NETCONF agents.

If the NETCONF session does not start:

- make sure the agent is reachable
  - try to 'ping' the agent and see if it responds
- make sure the SSH server is responding
  - try to login in to the agent using normal SSH login on port 22
- make sure a firewall is not blocking TCP port 830
  - try to connect to the NETCONF agent using the **--port=22** option
- make sure the netconf-subsystem is configured correctly in /etc/ssh/sshd_config there should be the proper configuration commands for NETCONF to work. For example, the netconfd agent configuration would look like this:

```
Port 22
Port 830
Subsystem   netconf     /usr/local/bin/netconf-subsystem
```

- make sure the netconfd agent is running. Use the unix 'ps' command, or check the netconfd log file, to make sure it is running.

```
ps -alx | grep netconf
(look for 1 'netconfd and N 'netconf-subsystem' -- 1 for
 each active session)
```

- make sure the user name is correct
  - This must be a valid user name on the system
- make sure the password is correct
  - This must be the valid password (in /etc/passwd or /etc/shadow) for the specified user name

If the NETCONF session stops responding:

- make sure the agent is still reachable
  - try to 'ping' the agent and see if it responds
- make sure the SSH server is still responding
  - try to login in to the agent using normal SSH login on port 22

If the NETCONF agent is not accepting a certain command:

- make sure the command (or parameters used in the command) is actually supported by the agent.
  - There may be features, when statements, or deviation statements that indicate the agent does not support the command or one or more of its parameters.
- make sure that access control configured on the agent is not blocking the command. The error-tag should be 'access-denied' in this case.

If the NETCONF agent is not returning the expected data in a <get> or <get-config> protocol operation::

- Make sure all the parameters are supported by the agent
  - the **:xpath** capability must be advertised by the agent to use the 'select' attribute in the <get> or <get-config> operations
  - the **:with-defaults** capability must be advertised by the agent to use the <with-defaults> parameter
- if using a filter, try to retrieve the data without a filter and see if it is there
- make sure that access control configured on the agent is not blocking the retrieval. There will not be any error reported in this case. The agent will simply skip over any unauthorized data, and leave it out of the <rpc-reply>.
- set the logging level to debug2 or higher, and look closely at the PDUs being sent to the agent. Set the display mode to a value other than 'plain' to make sure the correct namespaces are being used in the request.

If an <edit-config> operation is failing unexpectedly:

- make sure that access control configured on the agent is not blocking the request. The error-tag should be 'access-denied' in this case.
- make sure an unsupported parameter or parameter value is not used
  - <test-option> is not supported unless the **:validate** capability is advertised by the agent
  - <error-option> = 'rollback-on-error' is not supported unless the **:rollback-on-error** capability is advertised by the agent
- if the request contains an edit to a nested data structure, make sure the parent data structure(s) are in place as expected. The <default-operation> parameter is set to 'none' in the high level editing operations, so any data 'above' the edited data must already exist.
- set the logging level to debug2 or higher, and look closely at the PDUs being sent to the agent. Set the display mode to a value other than 'plain' to make sure the correct namespaces are being used in the request.

7.4 Command Reference

This section describes all the **yangcli** local and remote commands built-in when using the **netconfd** agent.

There may be more or less commands available, depending on the YANG modules actually loaded at the time.

The specific NETCONF capabilities needed are listed for each remote command. No capabilities are ever needed for a local command.

It is possible that other agents will support protocol operations that **netconfd** does not support. If yangcli has the YANG file available for the module, then it can be managed with the high level commands. Low-level commands can still be used with external data (e.g., @mydatafile.xml).

Any YANG rpc statement can be used as a remote **yangcli** command. Refer to the agent vendor documentation for details on the protocol operations, database contents, and notification definitions that they support.

7.4.1 CD

The **cd** command is used to change the current working directory.

cd command

Command type:	local

Default parameter:	dir
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	yangcli.yang

Command Parameters:

- **dir**
  - type: string
  - usage: mandatory
  - default: none
  - The '**dir**' string must contain a valid directory specification

Positive Response:

- the new current working directory is printed

Negative Response:

- an error message will be printed describing the error

Usage:

```
yangcli> cd ~/modules

Current working directory is /home/andy/modules

yangcli> cd --dir=$YANG_HOME

Current working directory is /home/andy/swdev/yangtools/trunk/netconf

yangcli>
```

7.4.2 CLOSE-SESSION

The **close-session** command is used to terminate the current NETCONF session. A NETCONF agent should always accept this command if it is valid, and not reject it due to access control enforcement or if the agent is in notification delivery mode.

close-session command

Command type:	remote
Default parameter:	none
Min parameters:	0
Max parameters:	0

Return type:	status
YANG file:	netconf.yang

Command Parameters:

- none

Positive Response:

- the session is terminated and the command prompt is changed to indicate idle mode

Negative Response:

- an <rpc-error> message will be printed describing the error

Usage:

```
yangcli andy@myagent> close-session

RPC OK Reply 2 for session 10:

yangcli>
```

Reference:

- RFC 4741, section 7.8

7.4.3 COMMIT

The **commit** command is used to save the edits in the <candidate> database into the <running> database. If there are no edits it will have no effect.

commit command

Command type:	remote
Default parameter:	none
Min parameters:	0
Max parameters:	2
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	:candidate
Capabilities optional:	:confirmed-commit

Command Parameters:

- **confirmed**
  - type: empty
  - usage: optional

    - default: none
    - capabilities needed: :confirmed-commit
    - This parameter requests a confirmed commit procedure. The agent will expect another **commit** command before the **confirm-timeout** time period expires.
- **confirm-timeout**
    - type: uint32 (range: 1 .. max)
    - usage: optional
    - default: 600 seconds
    - capabilities needed: :confirmed-commit
    - This is the number of seconds to request before the timeout.
      The '**confirmed**' leaf must also be present for this parameter to have any affect.

Positive Response:

- the session is terminated and the command prompt is changed to indicate idle mode

Negative Response:

- an <rpc-error> message will be printed describing the error

Usage:

```
yangcli andy@myagent> commit

RPC OK Reply 5 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 8.3.4

7.4.4 CONNECT

The **connect** command is used to start a session with a NETCONF agent.

If there already is a NETCONF session active, then an error message will be printed and the command will not be executed.

connect command

Command type:	remote
Default parameter:	agent
Min parameters:	3
Max parameters:	5
Return type:	status
YANG file:	yangcli.yang

Command Parameters:

- **agent**
  - type: inet:ip-address (string containing IP address or DNS name
  - usage: mandatory
  - default: previous agent used, if any, will be presented as the default, but not used automatically
  - This parameter specifies the agent address for the session.
- **password**
  - type: string (ncx:password)
  - usage: mandatory
  - default: previous password used, if any, will be presented as the default, but not used automatically
  - This parameter specifies the password string to use to establish the session. It will not be echoed in parameter mode or saved in the command history buffer.
- **port**
  - type: uint16
  - usage: optional
  - default: 830
  - This parameter specifies the TCP port number that should be used for the session.
- **timeout**
  - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
  - usage: optional
  - default: set to the **$$timeout** system variable, default 30 seconds
  - This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times out, but any late response will be dropped. A value of zero means no timeout should be used, and **yangcli** will wait forever for a response.
- **user**
  - type: string
  - usage: mandatory
  - default: previous user name used, if any, will be presented as the default, but not used automatically
  - This parameter specifies the user name to use for the session. This must be a valid user name on the NETCONF agent.

Positive Response:

- the session is started and the prompt changes to include the 'user@agent' string.

Negative Response:

- One or more error messages will be printed. Refer to the section on trouble-shooting NETCONF Session problems for more details.

Usage:

```
yangcli> connect myagent user=andy password=yangrocks

<startup screen printed>

yangcli andy@myagent>
```

7.4.5 COPY-CONFIG

The **copy-config** command is used to copy one entire NETCONF database to another location.

Not all possible parameter combinations will be supported by every agent. In fact, the NETCONF protocol does not require any parameters to be supported unless the **:startup** or **:url** capabilities is supported by the agent.

If the agent supports the :startup capability, then it must support:

```
yangcli andy@myagent> copy-config source=running target=startup
```

This is the standard way to save a snapshot of the current running configuration in non-volatile storage, if the agent has a separate startup database. If not, the agent will automatically save any changes to the running configuration to non-volatile storage.

copy-config command

Command type:	remote
Default parameter:	none
Min parameters:	2
Max parameters:	3
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	none
Capabilities optional:	:candidate :startup :url :with-defaults

Command Parameters:

- **source**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the source database for the copy operation.
  - container contents: 1 of N:
    - **candidate**
      - type: empty
      - capabilities needed: :candidate
    - **running**
      - type: empty
      - capabilities needed: none
    - **startup**
      - type: empty
      - capabilities needed: startup
    - **url**
      - type: yang:uri
      - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter
      - To enter this parameter, the interactive mode must be used. The shorthand mode (source=url) cannot be used, since this parameter contains a string.
- **target**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the target database for the copy operation.
  - container contents: 1 of N:
    - **candidate**
      - type: empty
      - capabilities needed: :candidate
    - **running**
      - type: empty
      - capabilities needed: none
    - **startup**

        - type: empty
        - capabilities needed: startup
    - **url**
        - type: yang:uri
        - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter.
        - To enter this parameter, the interactive mode must be used. The shorthand mode (target=url) cannot be used, since this parameter contains a string.
- **with-defaults**
    - type: enumeration (none report-all trim explicit)
    - usage: optional
    - default: none
    - capabilities needed: with-defaults
    - This parameter controls how nodes containing only default values are copied to the target database.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> copy-config source=candidate

Enter a number of the selected case statement:

  1: case candidate:
       leaf candidate
  2: case running:
       leaf running
  3: case startup:
       leaf startup
  4: case url:
       leaf url

Enter choice number [1 - 4]:
yangcli andy@myagent:copy-config> 4
```

```
Filling optional case /copy-config/input/target/config-source/url
Enter string value for leaf <url>:
yangcli andy@myagent:copy-config> file://configs/myconfig.xml

RPC OK Reply 12 for session 10:
yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.3

7.4.6 CREATE

The **create** command is a high-level <edit-config> operation. It is used to create some new nodes in the default target database.

A target node is specified (in 1 of 2 ways), and then the data structure is filled in. Only mandatory nodes will be filled in unless the **$$optional** system variable is set to 'true'.

Refer to the **fill** command for more details on interactive mode data structure completion.

create command

Command type:	remote
Default parameter:	target
Min parameters:	1
Max parameters:	5
Return type:	status
YANG file:	yangcli.yang
Capabilities needed:	:candidate or :writable-running

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'from-cli'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the create operation. It is either a user variable or from interactive input at the command line.
    - **varref**
      - type: string

        - usage: mandatory
        - default: none
        - This parameter specifies the name of the user variable to use for the target of the create operation. The parameter must exist (e.g., created with the **fill** command) or an error message will be printed.
    - **case from-cli** (not entered)
        - **target**
            - type: string
            - usage: mandatory
            - default: none
            - This parameter specifies the database target node of the create operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
        - **optional**
            - type: empty
            - usage: optional
            - default: controlled by **$$optional** system variable
            - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.
        - **value**
            - type: anyxml
            - usage: mandatory
            - default: none
            - This parameter specifies the value that should be used for the contents of the **target** parameter. If it is entered, then the interactive mode prompting for missing parameters will be skipped, if this parameter is complete (or all mandatory nodes present if the **$$optional** system variable is 'false'). For example, if the target is a leaf, then specifying this parameter will always cause the interactive prompt mode to be skipped.
- **timeout**
    - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
    - usage: optional
    - default: set to the **$$timeout** system variable, default 30 seconds
    - This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times out, but any late response will be dropped. A value of zero means no timeout should be used, and yangcli will wait forever for a response.

System Variables:

- **$$error-option**
  - The <error-option> parameter for the <edit-config> operation will be derived from this variable
- **$$test-option**
  - The <test-option> parameter for the <edit-config> operation will be derived from this variable

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> create varref=myvar

RPC OK Reply 10 for session 10:

yangcli andy@myagent> create /nacm/rules/dataRule \
(user will be prompted to fill in the dataRule contents)

RPC OK Reply 11 for session 10:

yangcli andy@myagent> create \
     target=/nacm/rules/dataRule[name='test rule']/comment \
     value="this test rule is temporary. Do not remove!"
(no user prompting; <edit-config> request sent right away)

RPC OK Reply 12 for session 10:
yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.2

7.4.7 CREATE-SUBSCRIPTION

The create-subscription command is used to start receiving notifications from the agent.

The :notifications capability must be supported by the agent to use this command.

Unless the :interleave capability is also supported by the agent, then only the **close-session** command can be used while notifications are being delivered.

create-subscription command

Command type:	remote
Default parameter:	none
Min parameters:	1

Max parameters:	4
Return type:	status
YANG file:	notifications.yang
Capabilities needed:	:notifications
Capabilities optional:	:interleave

Command Parameters:

- **stream**
  - type: string
  - usage: mandatory
  - default: 'NETCONF'
  - This parameter specifies the name of the notification stream for this subscription request. Only the 'NETCONF' stream is mandatory to implement. Any other stream contains vendor-specific content, and may not be fully supported, depending on the stream encoding.
- **filter**
  - type: anyxml (same as the <get> or <get-config> filter parameter)
  - usage: optional
  - default: none
  - This parameter specifies a boolean filter that should be applied to the stream. This is the same format as the standard <filter> element in RFC 4741, except that instead of creating a subset of the database for an <rpc-reply> PDU, the filter is used as a boolean test to send or drop each notification delivered from the agent.
    - If any nodes are left in the 'test response', the agent will send the entire notification.
    - If the result is empty after the filter is applied to the “test response”, then the agent will not send the notification at all.
    - It is possible that access control will either cause the a notification to be dropped entirely, or may be pruned and still delivered. The standard is not clear on this topic. The **netconfd** agent will prune any unauthorized payload from an eventType, but if the <eventType> itself is unauthorized, the entire notification will be dropped.
- **startTime**
  - type: yang:date-and-time
  - usage: optional
  - default: none
  - This parameter causes any matching replay notifications to be delivered by the agent, if notification replay is supported by the agent. A notification will match if its <eventTime> value is greater or equal to the value of this parameter.

- After all the replay notifications are delivered, the agent will send a <replayComplete eventType, indicating there are no more replay notifications that match the subscription request.

- **stopTime**
  - type: yang:date-and-time
  - usage: optional (not allowed unless startTime is also present)
  - default: none
  - This parameter causes any matching replay notifications to be delivered by the agent, if notification replay is supported by the agent. A notification will match if its <eventTime> value is less than the value of this parameter.
    - This parameter must be greater than the **startTime** parameter, or an error will be returned by the agent.
    - If this parameter is used, then the entire subscription will stop after this specified time, even if it is in the future. The <notificationComplete> eventType will be sent by the agent when this event occurs.
    - If this parameter is not used (but **startTime** is used), then the agent will continue to deliver 'live' notifications after the <replayComplete> eventType is sent by the agent.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> create-subscription

RPC OK Reply 13 for session 10:

yangcli andy@myagent> create-subscription \
     startTime=2009-01-01T00:00:00Z

RPC OK Reply 14 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 5277, section 2.1.1

7.4.8 DELETE

The **delete** command is a high-level <edit-config> operation. It is used to delete an existing subtree in the default target database.

A target node is specified, and then any missing key leafs (if any) within the data structure are filled in. If the target is a leaf-list, then the user will be prompted for the value of the leaf-list node to be deleted.

Refer to the **fill** command for more details on interactive mode data structure completion.

delete command

Command type:	remote
Default parameter:	target
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	yangcli.yang
Capabilities needed:	:candidate or :writable-running

Command Parameters:

- **target**
  - type: string
  - usage: mandatory
  - default: none
  - This parameter specifies the database target node of the delete operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.

System Variables:

- **$$error-option**
  - The <error-option> parameter for the <edit-config> operation will be derived from this variable
- **$$optional**
  - Controls whether optional descendant nodes will be filled into the **target** parameter contents
- **$$test-option**
  - The <test-option> parameter for the <edit-config> operation will be derived from this variable

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> delete /nacm/rules/dataRule \
(user will be prompted to fill in the dataRule 'name' key leaf)

RPC OK Reply 15 for session 10:

yangcli andy@myagent> delete \
     target=/nacm/rules/dataRule[name='test rule']/comment
(no user prompting; <edit-config> request sent right away)

RPC OK Reply 16 for session 10:
yangcli
```

Reference:

- RFC 4741, section 7.2

7.4.9 DELETE-CONFIG

The **delete-config** command is used to delete an entire NETCONF database.

Not all possible **target** parameter values will be supported by every agent. In fact, the NETCONF protocol does not require that any database be supported by this operation.

If the agent supports the :url capability, then it may support deletion of local file databases in this manner.:

delete-config command

Command type:	remote
Default parameter:	none
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	none
Capabilities optional:	:candidate :startup :url

Command Parameters:

- **target**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none

- This parameter specifies the name of the target database for the delete operation.
- container contents: 1 of N:
  - **startup**
    - type: empty
    - capabilities needed: startup
    - An agent may support this target
  - **url**
    - type: yang:uri
    - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter.
    - To enter this parameter, the interactive mode must be used. The shorthand mode (target=url) cannot be used, since this parameter contains a string.
    - An agent may support this parameter

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> delete-config target=startup

RPC OK Reply 17 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.4

7.4.10 DISCARD-CHANGES

The **discard-changes** command is used to delete any edits that exist in the <candidate> database, on the NETCONF agent. The agent will only accept this command if the :candidate capability is supported. If the <candidate> database is locked by another session, then this request will fail with an 'in-use' error.

discard-changes command

Command type:	remote

Default parameter:	none
Min parameters:	0
Max parameters:	0
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	:candidate

Command Parameters: none

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> discard-changes

RPC OK Reply 18 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 8.3.4.2

7.4.11 EDIT-CONFIG

The edit-config command allows a subset of a NETCONF database on the agent to be changed.

If the agent supports the **:url** capability, then it may support editing of local file databases.

If the agent supports the :candidate capability, then it will allow edits to the <candidate> database.

If the agent supports the :writable-running capability, it will support edits to the <running> database.

It is not likely that an agent will support the <candidate> and <running> database as targets at the same time, since changes to the <running> configuration would not be reflected in the <candidate> database, while it was being edited by a different session.

edit-config command

Command type:	remote

Default parameter:	none
Min parameters:	2
Max parameters:	5
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	:candidate or :writable-running
Capabilities optional:	:url :rollback-on-error :validate

Command Parameters:

- **default-operation**
  - type: enumeration (merge replace none)
  - usage: optional
  - default: merge
  - This parameter specifies which edit operation will be in effect at the start of the operation, before any **nc:operation** attribute is found.
    - The high-level edit operations provided by **yangcli** will set this parameter to 'none'. This is the safest value, since only subtrees that have an explicit **nc:operation** attribute in effect can possibly be altered by the command.
    - If the value is 'merge', then any missing nodes in the database will be automatically created as needed.
    - If the value is 'replace', then the target database will be pruned to match the edits, as needed. Only the data from the **config** parameter will remain if this value is used. (Use with extreme caution).
- **error-option**
  - type: enumeration (stop-on-error continue-on-error rollback-on-error
  - usage: optional
  - default: stop-on-error
  - This parameter specifies what the agent should do when an error is encountered.
    - The rollback-on-error value is only allowed if the **:rollback-on-error** capability is supported by the agent.
    - The standard is not clear what continue-on-error really means. It is suggested that this value not be used. It is possible that the agent will validate all input parameters before making any changes, no matter how this parameter is set.
- **choice edit-content** (not entered)
  - **config**
    - type: anyxml

        - usage: mandatory
        - default: none
        - This parameter specifies the subset of the database that should be changed. This is the most common way to edit a NETCONF agent database, since it is mandatory to support by all agents.
    - **url**
        - type: yang:uri
        - capabilities needed: **:url,** and the scheme used in the URL must be specified in the **:url** capability 'schemes' parameter.
        - To enter this parameter, the interactive mode must be used. The shorthand mode (target=url) cannot be used, since this parameter contains a string.
- **target**
    - type: container with 1 of N choice of leafs
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the target database for the edit operation.
    - container contents: choice: 1 of N:
        - **candidate**
            - type: empty
            - capabilities needed: :candidate
        - **running**
            - type: empty
            - capabilities needed: :writable-running
- **test-option**
    - type: enumeration (test-then-set set test-only)
    - usage: optional
    - default: set (unless **:validate** capability is supported, the 'test-then-set' is the default value)
    - This parameter specifies how the agent should test the **edit-content** parameter before using it.
        - If the value is 'set' (normal case), the agent will apply validation tests as needed for the individual data structures being edited
        - The value 'test-then-set' is only allowed if the :validate capability is supported by the agent. The agent will test if the entire database will be valid after the edits are made, before making any changes to the candidate configuration.
            - This mode is very resource intensive. Set this parameter to 'set' for better performance, and run the validation tests manually with the **validate** command.

- The value 'test-only' is not supported by all agents. It will be in the next version of the NETCONF protocol, but is non-standard at this time.
  - Use this value to check if a specific edit should succeed or not, allowing errors to be corrected before altering the database for real.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> edit-config target=candidate \
     default-operation=merge \
     test-option=test \
     error-option=stop-on-error \
     config=@myconfig.xml

RPC OK Reply 19 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.2

7.4.12 EVENTLOG

The **eventlog** command is used to view or clear all or part of the notification event log. This log will be empty if no well-formed notifications have been received from any agent.

The **eventlog show** command is used to display some event log entries.

The **eventlog clear** command is used to delete some event log entries.

If no parameters are entered, it is the same as entering 'eventlog show=-1'.

The event log is not automatically emptied when a session terminates, in case the session was dropped unexpectedly. New entries will be appended to the event log as new sessions and/or subscriptions are started.

eventlog command

Command type:	local
Default parameter:	show
Min parameters:	0

Max parameters:	3
Return type:	status
YANG file:	yangcli.yang

Command Parameters:

- **choice eventlog-action** (not entered):
  - type: choice of case 'show-case' or leaf 'clear'
  - usage: optional
  - default: show=-1 is used as the default if nothing entered
  - This parameter specifies the event log action that should be performed.
    - **clear**
      - type: int32 (-1 to clear all entries; 1 to max to delete N entries)
      - usage: optional
      - default: -1
      - This parameter specifies the maximum number of event log entries to be deleted, starting from the oldest entries in the event log. The value -1 means delete all the entries. Otherwise the value must be greater than zero, and up to that many entries will be deleted.
    - **case show-case** (not entered)
      - **choice help-mode** (not entered) (default choice is 'normal')
        - **brief**
          - type: empty
          - usage: optional
          - default: none
          - This parameter specifies that brief documentation mode should be used. The event log index, sequence ID, and <eventType> will be displayed in this mode.
        - **normal**
          - type: empty
          - usage: optional
          - default: none
          - This parameter specifies that normal documentation mode should be used. The event log index, <eventTime>, sequence ID, and <eventType> will be displayed in this mode.
        - **full**
          - type: empty
          - usage: optional

        - default: none
        - This parameter specifies that full documentation mode should be used. The event log index, <eventTime>, sequence ID, and <eventType> will be displayed in this mode. In addition, the entire contents of the notification PDU will be displayed, using the current **$$display-mode** setting.
    - **show**
        - type: int32 (-1 for all, 1 to max for N entries)
        - usage: optional
        - default: -1
        - This parameter specifies the number of event log entries that should be displayed. The value '-1' indicates all the entries should be displayed. Otherwise, the value must be greater than zero, indicating the number of entries to display.
    - **start**
        - type: uint32
        - usage: optional
        - default: 0
        - This parameter specifies the start position in the event log to start displaying entries. The first entry is number zero. Each time the event log is cleared, the numbering restarts.

System Variables:

- **$$display-mode**
    - The log entries printed when help-mode='full' are formatted according to the current value of this system variable.

Positive Response:

- the event log entries are printed or cleared as requested

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli andy@myagent> eventlog show=5 start=3

 [3]   [2009-07-10T02:21:10Z] (4) <sysSessionStart>
 [4]   [2009-07-10T02:23:14Z] (5) <sysSessionEnd>
 [5]   [2009-07-10T02:23:23Z] (6) <sysSessionStart>
 [6]   [2009-07-10T02:24:52Z] (7) <sysConfigChange>
 [7]   [2009-07-10T02:24:57Z] (8) <sysSessionEnd>

yangcli andy@myagent>
```

7.4.13 FILL

The **fill** command is used to create a user variable for reuse in other commands.

It is used in an assignment statement to create a variable from various sources.

If it is not used in an assignment statement, then the result will not be saved, so the command will have no effect in this case.

The value contents will mirror the subtree within the NETCONF database indicated by the target parameter. If not completely provided, then missing descendant nodes will be filled in interactively, by prompting for each missing node.

fill command

Command type:	local
Default parameter:	target
Min parameters:	2
Max parameters:	3
Return type:	data
YANG file:	yangcli.yang

Command Parameters:

- **optional**
  - type: empty
  - usage: optional
  - default: controlled by **$$optional** system variable
  - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.
- **target**
  - type: string
  - usage: mandatory
  - default: none
  - This parameter specifies the database target node of the create operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
- **value**
  - type: anyxml
  - usage: mandatory
  - default: none
  - This parameter specifies the content to use for the filled variable.

- If this parameter is not entered, then the user will be prompted interactively to fill in the **target** data structure.
- If a string is entered, then the target value being filled must be a leaf or leaf-list.
- If a variable reference is entered, then it will be used as the content, if the target value being filled is a leaf or a leaf-list.
- If the target value is a complex object, then the referenced variable must also be a complex object of the same type.
- An error will be reported if the global or local variable does not reference the same object type as the target parameter.

System Variables:

- **$$optional**
  - Controls whether optional descendant nodes will be filled into the **target** parameter contents

Positive Response:

- OK

Negative Response:

- An error message will be printed if errors are detected.

Output:

- data
  - type: anyxml
  - The data structure will mirror the requested target object.
  - The variable (if any) will retain the target object name and namespace so it can be used in other operations more easily. In the example below, the **$my_interface** local variable will have the module name 'interfaces' and name 'interface', when used in other commands such as **create** or **merge**.

Usage:

```
yangcli> $my-interface = fill \
     target=/interfaces/interface optional

     (user will be prompted to fill in all fields
      of the <interface> element)
   OK

yangcli>
```

7.4.14 GET

The **get** command is used to retrieve data from the agent.

get command

Command type:	remote
Default parameter:	none
Min parameters:	0
Max parameters:	2
Return type:	data
YANG file:	netconf.yang

Command Parameters:

- **filter**
  - type: anyxml
  - usage: optional
  - default: none
  - This parameter specifies a boolean filter that should be applied to the stream. Any data in the <running> database (or non-config data) that does not match the filter will be left out of the <rpc-reply> response.
    - If no filter is used, the agent will return the entire <running> database and all non-config data as well. This could be a lot of data, depending on the agent.
    - If the result is empty after the filter is applied to the available data, then the agent will send an empty <data> element in the <rpc-reply>
    - It is possible that access control will cause the <rpc-reply> to be pruned. The **netconfd** agent will silently prune any unauthorized payload from the <rpc-reply>.
- **with-defaults**
  - type: enumeration (none report-all trim explicit)
  - usage: optional
  - default: none
  - capabilities needed: with-defaults
  - This parameter controls how nodes containing only default values are returned in the <rpc-reply>.

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- **data**
  - type: anyxml

- This element will contain the requested data from the <running> database, or non-config data from the agent instrumentation.

Usage:

```
yangcli andy@myagent> get

RPC Data Reply 20 for session 10:

rpc-reply {
    data {
       …. data returned by the agent
    }
}

yangcli andy@myagent> get filter=@myfilter.xml

RPC Data Reply 21 for session 10:

rpc-reply {
    data {
    }
}

(the previous response will occur if the filter did not
 match anything or the agent access control filtered the
 entire response)

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.7

7.4.15 GET-CONFIG

The **get-config** command is used to retrieve configuration data from the agent.

get-config command

Command type:	remote
Default parameter:	none
Min parameters:	1
Max parameters:	3
Return type:	data
YANG file:	netconf.yang

Command Parameters:

- **filter**
  - type: anyxml
  - usage: optional
  - default: none
  - This parameter specifies a boolean filter that should be applied to the stream. Any data in the <running> database (or non-config data) that does not match the filter will be left out of the <rpc-reply> response.
    - If no filter is used, the agent will return the entire <running> database and all non-config data as well. This could be a lot of data, depending on the agent.
    - If the result is empty after the filter is applied to the available data, then the agent will send an empty <data> element in the <rpc-reply>
    - It is possible that access control will cause the <rpc-reply> to be pruned. The **netconfd** agent will silently prune any unauthorized payload from the <rpc-reply>.
- **source**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the source database for the retrieval operation.
  - container contents: 1 of N:
    - **candidate**
      - type: empty
      - capabilities needed: :candidate
    - **running**
      - type: empty
      - capabilities needed: none
    - **startup**
      - type: empty
      - capabilities needed: startup
    - **url**
      - type: yang:uri
      - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter
      - To enter this parameter, the interactive mode must be used. The shorthand mode (source=url) cannot be used, since this parameter contains a string.
- **with-defaults**

- type: enumeration (none report-all trim explicit)
- usage: optional
- default: none
- capabilities needed: with-defaults
- This parameter controls how nodes containing only default values are returned in the <rpc-reply>.

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- **data**
  - type: anyxml
  - This element will contain the requested data from the **source** database.

Usage:

```
yangcli andy@myagent> $my-config = get-config target=running

RPC Data Reply 22 for session 10:

rpc-reply {
    data {
       …. entire database returned by the agent
    }
}

yangcli andy@myagent> @saved-config.xml = get-config \
      filter=@myfilter.xml  \
      target=candidate

rpc-reply {
    data {
      … data requested by the filter
    }
}

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.1

7.4.16 GET-SCHEMA

The **get-schema** command is used to retrieve data model definition files from the agent. This is part of the NETCONF monitoring draft. The agent must support the **:schema-retrieval** capability to use this command.

If the agent reports a module or module version that **yangcli** cannot find in its local module library, then an error message will be printed. The **get-schema** command can then be used to retrieve the missing module from the agent.

The **ietf-netconf-state.yang** module includes a list of the schema supported by the agent, which can be retrieved from an agent that supports this module, such as **netconfd**.

```
yangcli andy@myagent> sget /ietf-netconf-state/schemas
```

The preceding command will return a <schemas> container with several <schema> child nodes. One example entry is shown below:

```
schemas {
    schema system 2009-06-04 YANG {
      identifier system
      version 2009-06-04
      format YANG
      namespace http://netconfcentral.com/ns/system
      location NETCONF
    }
}
```

The <identifier>, <version> and <format> leafs can be used as the corresponding parameter values for the **get-schema** command. See the example below for more details.

get-schema command

Command type:	remote
Default parameter:	none
Min parameters:	3
Max parameters:	3
Return type:	data
YANG file:	ietf-netconf-state.yang
Capabilities needed:	:schema-retrieval

Command Parameters:

- **identifier**

    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the module to retrieve.
        - Do not use any path specification of file extension; just the module name is entered.
        - The name is case-sensitive, and must be specified exactly as defined.
- **version**
    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the version of the module to retrieve.
        - For YANG modules, this will be the most recent revision date string defined in a module revision statement.
        - If any version is acceptable, or if the specific version is not known, then use the empty string.
- **format**
    - type: enumeration (XSD YANG RNG)
    - usage: mandatory
    - default: none
    - This parameter specifies the format of the module to be retrieved.
        - XSD: W3C REC REC-xmlschema-1-20041028
        - YANG: draft-ietf-netmod-yang
        - RNG: ISO/IEC 19757-2
        - **netconfd** only supports the YANG format

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- data
    - type: anyxml
    - **yangcli** will strip off this XML container if the command result is being saved to a text file. Only the YANG contents will be saved instead.

Usage:

```
yangcli andy@myagent> @notifications.yang = get-schema \
      identifier=notifications \
      version=2009-06-04 \
      format=YANG

RPC Data Reply 24 for session 10:

rpc-reply {
    data {
       …. entire notifications.yang contents
    }
}

(after retrieval, the module can be loaded locally
 with the mgrload command)

yangcli andy@myagent> mgrload notificications.yang

   OK

yangcli andy@myagent>
```

Reference:

- draft-ietf-netconf-monitoring-06.txt

7.4.17 HELP

The help command is used to print documentation to STDOUT.

If no session is active, then only help for the local commands and the standard NETCONF commands will be available.

If a NETCONF session is active, then the documentation shown will attempt to exactly match the capabilities of the agent.

If additional (i.e., augment generated) parameters are available, then they will be shown in the command output. If the agent does not implement some parameters (e.g., feature not supported) then these parameters will not be shown in the command output.

If the agent has modified an object with deviation statements, then the altered object will be shown.

The **ncx:hidden** extension suppresses the **help** command. If this extension is present in the YANG definition associated with the request, then no help will be available for that object or command.

help command

Command type:	local
Default parameter:	command

Min parameters:	3
Max parameters:	3
Return type:	data
YANG file:	ietf-netconf-state.yang
Capabilities needed:	:schema-retrieval

Command Parameters:

- **choice helptype** (not entered)
  - **command**
    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the command for which documentation is requested
  - **commands**
    - type: empty
    - usage: mandatory
    - default: none
    - This parameter will request documentation for all available commands
  - **notification**
    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the notification for which documentation is requested
  - **object**
    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the NETCONF database object for which documentation is requested.
      - Only top level objects are supported by this command.
      - Documentation for the entire object subtree will be printed, if the object is a container, choice, or list.
      - Documentation for nested objects is only available in parameter mode, using the escape commands for help ('?') and full help ('??')

    - **type**
        - type: string
        - usage: mandatory
        - default: none
        - This parameter specifies the name of the YANG typedef for which documentation is requested
        - Only top-level typedefs are supported by this command. Local typedefs within groupings, containers, or lists are not exportable in YANG.
- **choice help-mode** (not entered) (default choice is 'normal')
    - **brief**
        - type: empty
        - usage: optional
        - default: none
        - This parameter specifies that brief documentation mode should be used.
    - **normal**
        - type: empty
        - usage: optional
        - default: none
        - This parameter specifies that normal documentation mode should be used.
    - **full**
        - type: empty
        - usage: optional
        - default: none
        - This parameter specifies that full documentation mode should be used.

Positive Response:

- the agent prints the requested help text

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli> help help full

help
   Print the yangcli help text
   input
      default parameter: command
      choice helptype
```

```
         leaf command [NcxIdentifier]
            Show help for the specified command,
            also called an RPC method

         leaf commands [empty]
            Show info for all local commands

         leaf notification [NcxIdentifier]
            Show help for the specified notification

         leaf object [NcxIdentifier]
            Show help for the specified object

         leaf type [NcxIdentifier]
            Show help for the specified type

      choice help-mode
         leaf brief [empty]
            Show brief help text

         leaf normal [empty]
            Show normal help text

         leaf full [empty]
            Show full help text

yangcli andy@myagent> help notification sysConfigChange

notification sysConfigChange
   Generated when the <running> configuration is changed.
   leaf userName [string]

   leaf sessionId [SessionId]
      range: 1..max

   leaf remoteHost [ip-address]

   leaf target [string]

   leaf operation [EditOperationType] [d:merge]
```

```
        enum values: merge replace create delete

yangcli andy@svnserver>
```

7.4.18 HISTORY

The **history** command is used to show, clear, load, or save the command line history buffer.

Use the **recall** command to recall a previously executed command line, after getting the line number from the **history show** command.

All lines entered will be saved in the history buffer except an **ncx:password** value entered in parameter mode.

When **yangcli** starts, the command line history buffer is empty. If a history file was previously stored with the **history save** command, then it can be recalled into the buffer with the **history load** command.

The **history clear** command is used to delete the entire command line history buffer.

The numbering sequence for commands, starts from zero and keeps incremented until the program exits. If the history buffer is cleared, then the number sequence will continue, not start over at zero.

history command

Command type:	local
Default parameter:	show
Min parameters:	0
Max parameters:	2
Return type:	data
YANG file:	yangcli.yang

Command Parameters:

- **choice history-action** (not entered)
  - **clear**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that the history buffer should be cleared. Unless the contents have been saved with the **history save** command, there is no way to recover the cleared buffer contents after this command is executed.
  - **load**

        - type: string
        - usage: optional
        - default: $HOME/.yangcli_history
        - This parameter specifies a command history file, and causes the current command line history contents to be loaded from that file.
        - Special processing for this command allows the history file to be omitted in idle mode, even though the load parameter is not type 'empty'.

```
yangcli> history load
  same as:
yangcli> history load=$HOME/.yangcli_history
```

    - **save**
        - type: string
        - usage: optional
        - default: $HOME/.yangcli_history
        - This parameter specifies a command history file, and causes the current command line history contents to be saved to that file.
        - Special processing for this command allows the history file to be omitted in idle mode, even though the save parameter is not type 'empty'.
        -

```
yangcli> history save
  same as:
yangcli> history save=$HOME/.yangcli_history
```

    - **show**
        - type: int32 (-1 for all entries; 1..max for N entries)
        - usage: optional
        - default: -1
        - This parameter specifies the maximum number of history entries to show.
            - If no case is selected from this choice, then the command '**history show=-1**' will be used by default.
            - The **help-mode** choice parameter is only used with the **history show** command.
                - If the **--brief** or **--normal** modes are selected the the format will include the command number and the command line.
                - If the **--full** mode is selected, then the command data and time will also be printed.

- **choice help-mode** (not entered)
  This parameter is ignored unless the **history show** command is entered.
    - **brief**
        - type: empty

        - usage: optional
        - default: none
        - This parameter specifies that brief documentation mode should be used.
    - **normal**
        - type: empty
        - usage: optional
        - default: none
        - This parameter specifies that normal documentation mode should be used.
    - **full**
        - type: empty
        - usage: optional
        - default: none
        - This parameter specifies that full documentation mode should be used.

Positive Response:

- the requested history entries will be printed for the **history show** command
- all other commands will return OK

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli> history show=3 full
  [27]     2009-07-04 09:25:34 sget /system --nofill
  [28]     2009-07-04 09:34:17 @myconfig = get-config source=running
  [29]     2009-07-04 09:43:54 history show=3 full

yangcli> history save=~/my-temp-history-file
   OK
yangcli>
```

7.4.19 INSERT

The insert command is used to insert or move YANG list or leaf-list data into a NETCONF database. It is a high level command with utilizes the YANG 'insert' extensions to the NETCONF <edit-config> operation.

insert command

Command type:	remote
Default parameter:	target
Min parameters:	2
Max parameters:	7
Return type:	status
YANG file:	yangcli.yang

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'from-cli'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the insert operation. It is either a user variable or from interactive input at the command line.
    - **varref**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the name of the user variable to use for the target of the insert operation. The parameter must exist (e.g., created with the **fill** command) or an error message will be printed.
    - **case from-cli** (not entered)
      - **target**
        - type: string
        - usage: mandatory
        - default: none
        - This parameter specifies the database target node of the insert operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
      - **optional**
        - type: empty
        - usage: optional
        - default: controlled by **$$optional** system variable
        - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.
      - **value**

        - type: anyxml
        - usage: mandatory
        - default: none
        - This parameter specifies the value that should be used for the contents of the **target** parameter. If it is entered, then the interactive mode prompting for missing parameters will be skipped, if this parameter is complete (or all mandatory nodes present if the **$$optional** system variable is 'false'). For example, if the target is a leaf, then specifying this parameter will always cause the interactive prompt mode to be skipped.
- **edit-target**
    - type: string
    - usage: optional (must be present if the order parameter is set to 'before' or 'after').
    - default: none
    - This parameter specifies the value or key clause that should be used, as the list or leaf-list insertion point. It identifies the existing entry that the new entry will be inserted before or after, depending on the **order** parameter.
        - For a leaf-list, the edit-target contains the value of the target leaf-list node within the configuration being edited. E.g., edit-target='fred'.
        - For a list, the edit-target contains the key values of the target list node within the configuration being edited. E.g., edit-target=[name='fred'][zipcode=90210].
- **order**
    - type: enumeration (first last before after)
    - usage: optional
    - default: last
    - The insert order that should be used. If the value 'before' or 'after' is selected, then the **edit-target** parameter must also be present.
- **operation**
    - type: enumeration (create merge replace)
    - usage: optional
    - default: merge
    - This parameter specifies the **nc:operation** attribute value for the NETCONF <edit-config> operation. The insert operation is secondary to the NETCONF operation attribute.
- **timeout**
    - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
    - usage: optional
    - default: set to the **$$timeout** system variable, default 30 seconds

- This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times out, but any late response will be dropped. A value of zero means no timeout should be used, and yangcli will wait forever for a response.

System Variables:

- **$$error-option**
  - The <error-option> parameter for the <edit-config> operation will be derived from this variable
- **$$test-option**
  - The <test-option> parameter for the <edit-config> operation will be derived from this variable

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> insert varref=myvar order=first

RPC OK Reply 25 for session 10:

yangcli andy@myagent> insert /nacm/rules/dataRule \
            order=after \
            edit-target="[name='test-rule']"

(user will be prompted to fill in the dataRule contents)

RPC OK Reply 26 for session 10:

yangcli andy@myagent>
```

Reference:

- draft-ietf-netmod-yang-06

7.4.20 KILL-SESSION

The **kill-session** command is used to terminate a NETCONF session (other than the current session). All NETCONF implementations must support this command. It is needed sometimes to unlock a NETCONF database locked by a session that is idle or stuck.

If the **lock** command returns a 'lock-denied' <error-tag>, it will also include an <error-info> field called <session-id>. This is the session number that currently holds the requested lock. The same value should be used for the **session-id** parameter in this command, to terminate the session will the lock.

Note: this is an extreme measure, which should be used with caution.

kill-session command

Command type:	remote
Default parameter:	target
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	netconf.yang

Command Parameters:

- **session-id**
  - type: uint32 (range: 1.. max)
  - usage: mandatory
  - default: none
  - This parameter specifies the session number of the currently active NETCONF session that should be terminated.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> kill-session session-id=11

RPC OK Reply 27 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.9

7.4.21 LIST

This **list** command is used to display the commands, objects, and oids (object identifiers) that are available at the time.

The **list commands** command will display the local commands and the remote commands that are available in the current NETCONF session, or which have been loaded with the **mgrload** command.

The **list objects** command will display the top-level objects that are currently available in the current NETCONF session, or which have been loaded with the **mgrload** command.

The **list oids** command will display the object identifiers of the top-level objects that are currently available in the current NETCONF session, or which have been loaded with the **mgrload** command.

list command

Command type:	local
Default parameter:	none
Min parameters:	1
Max parameters:	3
Return type:	data
YANG file:	yangcli.yang

Command Parameters:

- **choice help-mode** (not entered)
  This parameter is ignored if the listtype choice is set to ' the **history show** command is entered.
  - **brief**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that brief documentation mode should be used.
  - **normal**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that normal documentation mode should be used.
  - **full**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that full documentation mode should be used.
- **choice 'listtype'** (not entered)
  - usage: mandatory
  - default: none
  - This parameter specifies the what type of data should be listed.

- **commands**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that the available commands should be listed.
    - If the **help-mode** is set to 'brief', then only the command names will be listed.
    - If the **help-mode** is set to 'normal', then the XML prefix and the command name will be listed.
    - If the **help-mode** is set to 'full', then the module name and the command name will be listed.
- **objects**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that the available top-level objects should be listed.
    - If the **help-mode** is set to 'brief', then only the object names will be listed.
    - If the **help-mode** is set to 'normal', then the XML prefix and the object name will be listed.
    - If the **help-mode** is set to 'full', then the module name and the object name will be listed.
- **oids**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that the available top-level object identifiers should be listed.
    - The **help-mode** parameter has no effect

- **module**
  - type: string
  - usage: optional
  - default: none
  - This parameter specifies a module name. If present then only information for the specified YANG module will be displayed.

Positive Response:

- the requested information will be printed

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli andy@myserver> list commands module=notifications

   ncEvent:create-subscription

yangcli andy@myserver>
```

7.4.22 LOAD

The **load** command is used to load a YANG module into the agent.

This command is only supported by the **netconfd** agent.

The YANG file must be available in the module search path for the agent.

Refer to the **netconfd** configuration section for more details on adding new YANG modules into the agent.

After using this command, the **mgrload** command may also be needed to keep the current session synchronized with the agent.

There is no **revision** parameter at this time, to load a specific revision of a module. It is complex for the agent instrumentation to support multiple module revisions at once. Currently, the agent developer decides which revision of the agent instrumentation is present.

The agent will return the revision date of the module that was loaded (or already present).

load command

Command type:	remote
Default parameter:	module
Min parameters:	1
Max parameters:	1
Return type:	data
YANG file:	netconfd.yang

Command Parameters:

- module
  - type: string (length 1 .. 63)
  - usage: mandatory
  - default: none

- This parameter specifies the name of the module to load.

Positive Response:

- the agent returns <mod-revision>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> load toaster

RPC Data Reply 27 for session 10:

rpc-reply {
   mod-revision 2009-06-23
}

yangcli andy@myagent>
```

7.4.23 LOCK

The **lock** command is used to request a global lock on a NETCONF database. It is used, along with the **unlock** command, to obtain exclusive write access to the NETCONF agent.

The scope of a lock command is the lifetime of the session that requested the lock. This means that if the session that owns the lock is dropped for any reason, all the locks it holds will be released immediately by the agent.

The use of database locks is optional in NETCONF, but it must be implemented by every agent. It is strongly suggested that locks be used if multiple managers are likely to log into the particular NETCONF agent.

One or more locks may be needed to execute a transaction safely:

- If the **:writable-running** capability is supported, then a lock on the <running> database is needed. This database can be locked at any time.
- If the **:startup** capability is supported, then a lock on the <startup> database is needed. This database can be locked at any time.
- If the **:candidate** capability is supported, then a lock on the <candidate> database is needed. A lock on the <running> database is also needed.
  - The <candidate> database can only be locked if there are no active edits in it.
  - The **discard-changes** command may be needed to clear a <candidate> database that has been left edited by a session that terminated unexpectedly.
  - Note: There is no way in NETCONF to tell the difference between an actively edited <candidate> database and an 'abandoned' <candidate> database. The agent will almost never clear the <candidate> database. It will only clear any locks held. Use the discard-changes command (for other session's edits) with extreme caution.

lock command

Command type:	remote
Default parameter:	none
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	none
Capabilities optional:	:candidate :startup :url

Command Parameters:

- **target**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the target database to be locked.
  - container contents: 1 of N:
    - **candidate**
      - type: empty
      - capabilities needed: :candidate
    - **running**
      - type: empty
      - capabilities needed: none
    - **startup**
      - type: empty
      - capabilities needed: startup
    - **url**
      - type: yang:uri
      - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter.
      - To enter this parameter, the interactive mode must be used. The shorthand mode (target=url) cannot be used, since this parameter contains a string.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.
  - If the <error-tag> is 'lock-denied' then the <error-info> will contain a <session-id> leaf. This identifies the session number of the current lock holder.

Usage:

```
yangcli andy@myagent> lock target=candidate

RPC OK Reply 29 for session 10:

yangcli andy@myagent>
```

7.4.24 MERGE

The **merge** command is a high-level <edit-config> operation. It is used to merge some new nodes into the default target database.

A target node is specified (in 1 of 2 ways), and then the data structure is filled in. Only mandatory nodes will be filled in unless the **$$optional** system variable is set to 'true'.

Refer to the **fill** command for more details on interactive mode data structure completion.

merge command

Command type:	remote
Default parameter:	target
Min parameters:	1
Max parameters:	5
Return type:	status
YANG file:	yangcli.yang
Capabilities needed:	:candidate or :writable-running

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'from-cli'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the merge operation. It is either a user variable or from interactive input at the command line.

- **varref**
  - type: string
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the user variable to use for the target of the merge operation. The parameter must exist (e.g., created with the **fill** command) or an error message will be printed.
- **case from-cli** (not entered)
  - **target**
    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the database target node of the merge operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
  - **optional**
    - type: empty
    - usage: optional
    - default: controlled by **$$optional** system variable
    - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.
  - **value**
    - type: anyxml
    - usage: mandatory
    - default: none
    - This parameter specifies the value that should be used for the contents of the **target** parameter. If it is entered, then the interactive mode prompting for missing parameters will be skipped, if this parameter is complete (or all mandatory nodes present if the **$$optional** system variable is 'false'). For example, if the target is a leaf, then specifying this parameter will always cause the interactive prompt mode to be skipped.

- **timeout**
  - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
  - usage: optional
  - default: set to the **$$timeout** system variable, default 30 seconds

- This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times out, but any late response will be dropped. A value of zero means no timeout should be used, and **yangcli** will wait forever for a response.

System Variables:

- **$$error-option**
  - The <error-option> parameter for the <edit-config> operation will be derived from this variable
- **$$test-option**
  - The <test-option> parameter for the <edit-config> operation will be derived from this variable

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> merge \
     target=/nacm/rules/moduleRule[moduleName='netconf']\
     [allowedRights='read write']/allowedGroup \
     value=ncx:guest

(no user prompting; <edit-config> request sent right away)

RPC OK Reply 31 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.2

7.4.25 MGRLOAD

The **mgrload** command is used to load a YANG module into **yangcli**.

The YANG file must be available in the module search path for **yangcli**.

mgrload command

Command type:	local
Default parameter:	module
Min parameters:	1
Max parameters:	1

Return type:	status
YANG file:	yangcli.yang

Command Parameters:

- module
  - type: string (length 1 .. 63)
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the module to load.

Positive Response:

- OK

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli> mgrload toaster

Load module toaster OK

yangcli>
```

7.4.26 NO-OP

The **no-op** command is used to test agent message processing response times, by providing a baseline response time to do nothing except return <ok/>.

It can also be used as an application-level keep-alive to prevent proxy idle timeout or agent idle timeout problems from occurring.

This command is only supported by the **netconfd** agent. The agent will simply respond 'OK', if the request is well-formed.

no-op command

Command type:	remote
Default parameter:	none
Min parameters:	0
Max parameters:	0
Return type:	status
YANG file:	netconfd.yang

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> no-op

RPC OK Reply 31 for session 10:

yangcli andy@myagent>
```

7.4.27 PWD

The **pwd** command is used to print the current working directory.

pwd command

Command type:	local
Default parameter:	none
Min parameters:	0
Max parameters:	0
Return type:	status
YANG file:	yangcli.yang

Positive Response:

- the current working directory is printed to the log or STDOUT

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli> pwd

Current working directory is /home/andy

yangcli>
```

7.4.28 QUIT

The **quit** command is used to terminate the **yangcli** program.

If a NETCONF session is in progress, it will be dropped without sending a **close-session** command first. This should be taken into account if the agent reports dropped TCP connections as an error.

quit command

Command type:	local
Default parameter:	none
Min parameters:	0
Max parameters:	0
Return type:	none
YANG file:	yangcli.yang

Positive Response:

- The program terminates; no response will be printed.

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli> quit

andy@myworkstation>
```

7.4.29 RECALL

The **recall** command is used to recall a previously entered command line from the command line history buffer.

A command is recalled by its command ID number. Use the **history show** command to see the contents of the command line history buffer.

recall command

Command type:	local
Default parameter:	index
Min parameters:	1
Max parameters:	1
Return type:	data
YANG file:	yangcli.yang

Positive Response:

- The specified command line is recalled into the current command line.

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli> recall 7

yangcli>sget /system
```

7.4.30 REPLACE

The **replace** command is a high-level <edit-config> operation. It is used to replace an existing subtree with some new nodes, in the default target database.

Only the subtree indicated by the **target** or **varref** parameter will be replaced.

A target node is specified (in 1 of 2 ways), and then the data structure is filled in. Only mandatory nodes will be filled in unless the **$$optional** system variable is set to 'true'.

Refer to the **fill** command for more details on interactive mode data structure completion.

replace command

Command type:	remote
Default parameter:	target
Min parameters:	1
Max parameters:	5
Return type:	status
YANG file:	yangcli.yang
Capabilities needed:	:candidate or :writable-running

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'from-cli'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the replace operation. It is either a user variable or from interactive input at the command line.
    - **varref**

        - type: string
        - usage: mandatory
        - default: none
        - This parameter specifies the name of the user variable to use for the target of the replace operation. The parameter must exist (e.g., created with the **fill** command) or an error message will be printed.
    - **case from-cli** (not entered)
        - **target**
            - type: string
            - usage: mandatory
            - default: none
            - This parameter specifies the database target node of the replace operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
        - **optional**
            - type: empty
            - usage: optional
            - default: controlled by **$$optional** system variable
            - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.
        - **value**
            - type: anyxml
            - usage: mandatory
            - default: none
            - This parameter specifies the value that should be used for the contents of the **target** parameter. If it is entered, then the interactive mode prompting for missing parameters will be skipped, if this parameter is complete (or all mandatory nodes present if the **$$optional** system variable is 'false'). For example, if the target is a leaf, then specifying this parameter will always cause the interactive prompt mode to be skipped.
- **timeout**
    - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
    - usage: optional
    - default: set to the **$$timeout** system variable, default 30 seconds
    - This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times

out, but any late response will be dropped. A value of zero means no timeout should be used, and **yangcli** will wait forever for a response.

System Variables:

- **$$error-option**
  - The <error-option> parameter for the <edit-config> operation will be derived from this variable
- **$$test-option**
  - The <test-option> parameter for the <edit-config> operation will be derived from this variable

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> replace \
      target=/nacm/rules/moduleRule[moduleName='netconfd']\
      [allowedRights='exec']

(user prompted to fill in specified <moduleRule> element)

RPC OK Reply 31 for session 10:

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.2

7.4.31 RESTART

The **restart** command is used to restart the NETCONF agent.

This command is only supported by the **netconfd** agent.

The default access control configuration for **netconfd** will not allow any user except the designated 'superuser' account to invoke this command. Refer to the **netconfd** user manual for details on configuring access control.

restart command

Command type:	remote
Default parameter:	none
Min parameters:	0

Max parameters:	0
Return type:	status
YANG file:	netconfd.yang

Positive Response:

- the agent will drop all sessions and restart

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> restart

ses: session 10 shut by remote peer

yangcli>
```

7.4.32 RUN

The run command is used to run a **yangcli** script.

Refer to the section on scripts for details on how to write a script.

The script name is the only mandatory parameter. There are 9 generic parameters (named P1 to P9) that can be used to pass string parameters to scripts. Within a script, these are referenced with local variables ($1 to $9).

The run command can be used within a script.

Scripts do not return any status or data at this time.

The run command can appear inside a script, starting a new run level. An error will occur if a loop occurs or too many nest levels are used. Up to 64 run levels are supported in **yangcli**.

run command

Command type:	local
Default parameter:	script
Min parameters:	1
Max parameters:	10
Return type:	status
YANG file:	yangcli.yang

Command Parameters:

- **P1, P2, P3, P4, P5, P6, P7, P8, P9**
  - type: string
  - usage: optional
  - default: none
  - These parameters provide a generic string parameter passing mechanism for each script invocation, similar to unix shell scripts. Within the script, the parameters **$1**, **$2, $3**, **$4**, **$5**, **$6**, **$7**, **$8** and **$9** will be non-NULL only if the corresponding '**Pn**' parameter was set in the script invocation.
- **script**
  - type: string
  - usage: mandatory
  - default: none
  - This parameter specifis the name of the script to be run. If a path specification is included, then it will be used. Otherwise the current script search path will be used to find the script.

System Variables:

- **$$runpath**
  - Controls where **yangcli** will look for files specified in the **script** parameter.

Positive Response:

- the specified script will be executed

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error, if any remote commands are contained in the script.

Usage:

```
yangcli> run connect P1=yangrocks

runstack: Starting level 1 script /home/andy/scripts/connect

(commands and responses are printed as if they were typed)

runstack: Ending level 1 script /home/any/scripts/connect
yangcli andy@myserver>
```

7.4.33 SAVE

The **save** command is used to save NETCONF database edits to non-volatile storage, on the agent. It is a pseudo-command, mapped to other remote commands depending on which database target is supported by the agent (<candidate> or <running>).

The **save** command usually maps to either the **commit** or the **copy-config** command. Refer to the section on NETCONF sessions for more details.

save command

Command type:	remote
Default parameter:	none
Min parameters:	0
Max parameters:	0
Return type:	none
YANG file:	yangcli.yang

Positive Response:

- The agent returns one or two <ok/> responses.

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myserver> save

RPC OK Reply 34 on session 10

yangcli andy@myserver>
```

7.4.34 SGET

The **sget** command is used to invoke a NETCONF <get> operation with a subtree filter.

A target node is specified (in 1 of 2 ways), and then the data structure is filled in. Only mandatory nodes will be filled in unless the **$$optional** system variable is set to 'true'. This step can be skipped completely with the **nofill** parameter.

Refer to the **fill** command for more details on interactive mode data structure completion.

sget command

Command type:	remote
Default parameter:	target
Min parameters:	1
Max parameters:	5
Return type:	data
YANG file:	yangcli.yang

Capabilities needed:	none
Capabilities optional:	:with-defaults

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'from-cli'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the subtree filter target of the <get> operation. It is either a user variable or from interactive input at the command line.
    - **varref**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the name of the user variable to use for the subtree filter target of the <get> operation. The parameter must exist (e.g., created with the **fill** command) or an error message will be printed.
    - **case from-cli** (not entered)
      - **target**
        - type: string
        - usage: mandatory
        - default: none
        - This parameter specifies the database target node of the <get> operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
          - The escape command ('?s') can be used in parameter mode to skip a parameter completely.
          - Entering the empty string for a parameter will cause a subtree selection node to be created instead of a content match node
      - **optional**
        - type: empty
        - usage: optional
        - default: controlled by **$$optional** system variable
        - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.

        - **value**
            - type: anyxml
            - usage: mandatory
            - default: none
            - This parameter specifies the value that should be used for the contents of the **target** parameter. If it is entered, then the interactive mode prompting for missing parameters will be skipped, if this parameter is complete (or all mandatory nodes present if the **$$optional** system variable is 'false'). For example, if the target is a leaf, then specifying this parameter will always cause the interactive prompt mode to be skipped.
- **nofill**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies the that no interactive prompting to fill in the contents of the specified subtree filter target will be done.
        - This causes the entire selected subtree to be requested in the filter.
        - yangcli will attempt to figure out if there can be multiple instances of the requested subtree. If not, then **nofill** will be the default for that target parameter.
- **with-defaults**
    - type: enumeration (none report-all trim explicit)
    - usage: optional
    - default: none
    - capabilities needed: with-defaults
    - This parameter controls how nodes containing only default values are returned in the <rpc-reply>.

System Variables:

- **$$timeout**
    - The response timeout interval will be derived from this system variable.
- **$$with-defaults**
    - The <with-defaults> parameter for the <get> operation will be derived from this system variable.

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- **data**
  - type: anyxml
  - This element will contain the requested data from the <running> database or non-config data structures.

Usage:

```
yangcli andy@myagent> sget sytem

Filling container /system:
RPC Data Reply 32 for session 10:

rpc-reply {
   data {
      system {
         sysName myserver.localdomain
         sysCurrentDateTime 2009-07-06T02:24:19Z
         sysBootDateTime 2009-07-05T19:22:28Z
      }
   }
}

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.7

7.4.35 SGET-CONFIG

The **sget-config** command is used to invoke a NETCONF <get-config> operation with a subtree filter.

A target node is specified (in 1 of 2 ways), and then the data structure is filled in. Only mandatory nodes will be filled in unless the **$$optional** system variable is set to 'true'. This step can be skipped completely with the **nofill** parameter.

Refer to the **fill** command for more details on interactive mode data structure completion.

sget-config command

Command type:	remote
Default parameter:	target
Min parameters:	2
Max parameters:	6

Return type:	data
YANG file:	yangcli.yang
Capabilities needed:	none
Capabilities optional:	:candidate :startup :url :with-defaults

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'from-cli'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the subtree filter target of the <get> operation. It is either a user variable or from interactive input at the command line.
    - **varref**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the name of the user variable to use for the subtree filter target of the <get> operation. The parameter must exist (e.g., created with the **fill** command) or an error message will be printed.
    - **case from-cli** (not entered)
      - **target**
        - type: string
        - usage: mandatory
        - default: none
        - This parameter specifies the database target node of the <get> operation. This is an **ncx:schema-instance** string, so any instance identifier, or absolute path expression, or something in between, is accepted.
          - The escape command ('?s') can be used in parameter mode to skip a parameter completely.
          - Entering the empty string for a parameter will cause a subtree selection node to be created instead of a content match node
      - **optional**
        - type: empty

        - usage: optional
        - default: controlled by **$$optional** system variable
        - This parameter controls whether optional nodes within the target will be filled in. It can be used to override the **$$optional** system variable, when it is set to 'false'.
    - **value**
        - type: anyxml
        - usage: mandatory
        - default: none
        - This parameter specifies the value that should be used for the contents of the **target** parameter. If it is entered, then the interactive mode prompting for missing parameters will be skipped, if this parameter is complete (or all mandatory nodes present if the **$$optional** system variable is 'false'). For example, if the target is a leaf, then specifying this parameter will always cause the interactive prompt mode to be skipped.
- **nofill**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies the that no interactive prompting to fill in the contents of the specified subtree filter target will be done.
        - This causes the entire selected subtree to be requested in the filter.
        - yangcli will attempt to figure out if there can be multiple instances of the requested subtree. If not, then **nofill** will be the default for that target parameter.
- **source**
    - type: container with 1 of N choice of leafs
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the source database for the retrieval operation.
    - container contents: 1 of N:
        - **candidate**
            - type: empty
            - capabilities needed: :candidate
        - **running**
            - type: empty
            - capabilities needed: none

        - **startup**
            - type: empty
            - capabilities needed: startup
        - **url**
            - type: yang:uri
            - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter
            - To enter this parameter, the interactive mode must be used. The shorthand mode (source=url) cannot be used, since this parameter contains a string.
- **with-defaults**
    - type: enumeration (none report-all trim explicit)
    - usage: optional
    - default: none
    - capabilities needed: with-defaults
    - This parameter controls how nodes containing only default values are returned in the <rpc-reply>.

System Variables:

- **$$timeout**
    - The response timeout interval will be derived from this system variable.
- **$$with-defaults**
    - The <with-defaults> parameter for the <get-config> operation will be derived from this system variable.

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- **data**
    - type: anyxml
    - This element will contain the requested data from the database indicated by the **source** parameter.

Usage:

```
yangcli andy@myagent> sget-config /nacm/rules/dataRule \
     source=candidate \
     nofill

RPC Data Reply 35 for session 10:

rpc-reply {
```

```
    data {
      nacm {
          rules {
             dataRule nacm-tree {
                name nacm-tree
                path /nacm:nacm
                allowedRights {
                   read
                   write
                }
                allowedGroup nacm:admin
                comment 'access to nacm config'
             }
          }
       }
    }
}

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.1

7.4.36 SHOW

The **show** command is used to show program status and YANG details that may be needed during management of a NETCONF agent.

There are several variants of the **show** command:

- The **show global** command prints the contents of one global user variable.
- The **show globals** command prints a summary or the contents of all the global user variables.
- The **show local** command prints the contents of one local user variable.
- The **show locals** command prints a summary or the contents of all the local user variables.
- The **show module** command prints meta information or help text for one YANG module.
- The **show modules** command prints meta information for all the currently available YANG modules. IF a session is active, this will be the list of modules the agent reported, plus any modules loaded with the **mgrload** command.

- The **show objects** command prints the available objects or help text for the available objects.
- The **show var** command prints the contents of the specified variable.
- The **show vars** command prints a summary or the contents of all the program variables.
- The **show version** command prints the **yangcli** version number

show command

Command type:	local
Default parameter:	none
Min parameters:	1
Max parameters:	2
Return type:	data
YANG file:	yangcli.yang

Command Parameters:

- **choice help-mode** (not entered) (default choice is 'normal')
  - **brief**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that brief documentation mode should be used.
  - **normal**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that normal documentation mode should be used.
  - **full**
    - type: empty
    - usage: optional
    - default: none
    - This parameter specifies that full documentation mode should be used.
- **choice showtype** (not entered)
  - mandatory 1 of N choice for the sub-command for the **show** command
    - **global**

- type: string
- usage: mandatory
- default: none
- This parameter specifies the name of the global variable to show information for.
  - If **help-mode** is 'brief', then the name, variable object, and type of the global variable object will be printed.
  - If **help-mode** is 'normal' or 'full', then the contents of the specified global variable will be printed.

- **globals**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that information for all global variables should be printed.
    - If **help-mode** is 'brief', then the name, variable object, and type of each global variable object will be printed.
    - If **help-mode** is 'normal' or 'full', then the contents of each global variable will be printed.
- **local**
  - type: string
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the local variable to show information for.
    - If **help-mode** is 'brief', then the name, variable object, and type of the local variable object will be printed.
    - If **help-mode** is 'normal' or 'full', then the contents of the specified local variable will be printed.
- **locals**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that information for all local variables should be printed.
    - If **help-mode** is 'brief', then the name, variable object, and type of each local variable object will be printed.

        - If **help-mode** is 'normal' or 'full', then the contents of each local variable will be printed.
- **module**
    - type: string
    - usage: mandatory
    - default: none
    - This parameter specifies the name of the module to show information for.
        - If **help-mode** is 'brief' or 'normal', then the name, version, namespace, and source of the module will be printed.
        - If **help-mode** is 'full', then the module level help text for the specified module will be printed.
- **modules**
    - type: empty
    - usage: mandatory
    - default: none
    - This parameter specifies that information for all available modules should be printed:
        - If **help-mode** is 'brief', then the name of each module will be printed.
        - If **help-mode** is 'normal', then the XML prefix, name, and revision date of each module will be printed.
        - If **help-mode** is 'full', then the name, revision date, prefix, namespace, and source of each module will be printed.
- **objects**
    - type: empty
    - usage: mandatory
    - default: none
    - This parameter specifies that information for all available database objects should be printed:
        - If **help-mode** is 'brief', then the module name and name of each object will be printed.
        - If **help-mode** is 'normal', then the YANG object type, object name, and YANG base type will be printed.

          If **help-mode** is 'full', then brief help text will be printed for every object.
- **var**
    - type: string
    - usage: mandatory
    - default: none

- This parameter specifies the name of any variable to show information for.
  - If **help-mode** is 'brief', then the name, variable object, and type of the variable object will be printed.
  - If **help-mode** is 'normal' or 'full', then the contents of the specified variable will be printed.

- **vars**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that information for all variables should be printed. This includes all CLI, read-only system, read-write system, global user, and local user variables.
    - If **help-mode** is 'brief', then the name, variable object, and type of each variable object will be printed.
    - If **help-mode** is 'normal' or 'full', then the contents of each variable will be printed.
- **version**
  - type: empty
  - usage: mandatory
  - default: none
  - This parameter specifies that the yangcli program version string should be printed.
    - The **help-mode** parameter has no affect in this mode.

Positive Response:

- the agent prints the requested information

Negative Response:

- An error message will be printed if errors are detected.

Usage:

```
yangcli andy@myagent> show modules

  nc:netconf/2009-04-10
  inet:ietf-inet-types/2009-05-13
  ns:ietf-netconf-state/2009-03-03
  wd:ietf-with-defaults/2009-04-10
  yang:ietf-yang-types/2009-05-13
  nacm:nacm/2009-05-13
  manageEvent:nc-notifications/2008-07-14
  ncx:ncx/2009-06-12
```

```
  ncxapp:ncx-app-common/2009-04-10
  nt:ncxtypes/2008-07-20
  nd:netconfd/2009-05-28
  ncEvent:notifications/2008-07-14
  sys:system/2009-06-04
  t:test/2009-06-10

yangcli andy@myagent>
```

7.4.37 SHUTDOWN

The **shutdown** command is used to shut down the NETCONF agent.

This command is only supported by the **netconfd** agent.

The default access control configuration for **netconfd** will not allow any user except the designated 'superuser' account to invoke this command. Refer to the **netconfd** user manual for details on configuring access control.

shutdown command

Command type:	remote
Default parameter:	none
Min parameters:	0
Max parameters:	0
Return type:	status
YANG file:	netconfd.yang

Positive Response:

- the agent will drop all sessions and terminate.

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> shutdown

ses: session 10 shut by remote peer

yangcli>
```

7.4.38 UNLOCK

The **unlock** command is used to request a global lock on a NETCONF database. It is used, along with the **lock** command, to obtain exclusive write access to the NETCONF agent.

unlock command

Command type:	remote
Default parameter:	none
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	none
Capabilities optional:	:candidate :startup :url

Command Parameters:

- **target**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the target database to be locked.
  - container contents: 1 of N:
    - **candidate**
      - type: empty
      - capabilities needed: :candidate
    - **running**
      - type: empty
      - capabilities needed: none
    - **startup**
      - type: empty
      - capabilities needed: startup
    - **url**
      - type: yang:uri
      - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter.

- To enter this parameter, the interactive mode must be used. The shorthand mode (target=url) cannot be used, since this parameter contains a string.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> unlock target=candidate

RPC OK Reply 38 for session 10:

yangcli andy@myagent>
```

7.4.39 VALIDATE

The **validate** command is used to check if a complete NETCONF database is valid or not.

The **:validate** capability must be supported by the agent to use this command.

validate command

Command type:	remote
Default parameter:	none
Min parameters:	1
Max parameters:	1
Return type:	status
YANG file:	netconf.yang
Capabilities needed:	:validate
Capabilities optional:	:candidate :startup :url

Command Parameters:

- **target**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the target database to be validated.

- container contents: 1 of N:
  - **candidate**
    - type: empty
    - capabilities needed: :candidate
  - **running**
    - type: empty
    - capabilities needed: none
  - **startup**
    - type: empty
    - capabilities needed: startup
  - **url**
    - type: yang:uri
    - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter.
    - To enter this parameter, the interactive mode must be used. The shorthand mode (target=url) cannot be used, since this parameter contains a string.
  - **config**
    - type: anyxml
    - capabilities
    - This parameter represents an entire database, using the in-line config form, similar to the **edit-config** command, except this config parameter contains no nc:operation attributes and must be a complete database, not a subset.

Positive Response:

- the agent returns <ok/>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Usage:

```
yangcli andy@myagent> validate source=candidate

RPC OK Reply 36 for session 10:

yangcli andy@myagent>
```

7.4.40 XGET

The **xget** command is used to invoke a NETCONF <get> operation with an XPath filter.

xget command

Command type:	remote
Default parameter:	select
Min parameters:	1
Max parameters:	3
Return type:	data
YANG file:	yangcli.yang
Capabilities needed:	none
Capabilities optional:	:with-defaults

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'select'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the XPath filter target of the <get> operation. It is an XPath expression, either contained in a user variable or directly from the **select** parameter.
    - **varref**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the name of the user variable the contains the XPath expression for the XPath filter in the <get> operation. The parameter must exist (e.g., created with an assignment statement) or an error message will be printed.
    - **select**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the XPath expression to use for the XPath filter in the <get> operation.
- **timeout**
  - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
  - usage: optional
  - default: set to the **$$timeout** system variable, default 30 seconds

- This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times out, but any late response will be dropped. A value of zero means no timeout should be used, and **yangcli** will wait forever for a response.

- **with-defaults**
  - type: enumeration (none report-all trim explicit)
  - usage: optional
  - default: none
  - capabilities needed: with-defaults
  - This parameter controls how nodes containing only default values are returned in the <rpc-reply>.

System Variables:

- **$$timeout**
  - The response timeout interval will be derived from this system variable.
- **$$with-defaults**
  - The <with-defaults> parameter for the <get> operation will be derived from this system variable.

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- **data**
  - type: anyxml
  - This element will contain the requested data from the <running> database or non-config data structures.

Usage:

```
yangcli andy@myagent> xget //name

RPC Data Reply 42 for session 10:

rpc-reply {
   data {
      nacm {
         rules {
            dataRule nacm-tree {
               name nacm-tree
            }
         }
```

```
        }
        xpath.1 {
           name barney
        }
        ietf-netconf-state {
           datastores {
              datastore {
                 name {
                    candidate
                 }
              }
           }
        }
        netconf {
           streams {
              stream NETCONF {
                 name NETCONF
              }
           }
        }
     }
     }
}

yangcli andy@myagent>
```

Reference:

- RFC 4741, section 7.7

7.4.41 XGET-CONFIG

The **xget-config** command is used to invoke a NETCONF <get-config> operation with an XPath filter.

xget-config command

Command type:	remote
Default parameter:	select
Min parameters:	2

Max parameters:	4
Return type:	data
YANG file:	yangcli.yang
Capabilities needed:	none
Capabilities optional:	:candidate :startup :url :with-defaults

Command Parameters:

- **choice 'from'** (not entered)
  - type: choice of case 'varref' or case 'select'
  - usage: mandatory
  - default: none
  - This parameter specifies the where **yangcli** should get the data from, for the XPath filter target of the <get-config> operation. It is an XPath expression, either contained in a user variable or directly from the **select** parameter.
    - **varref**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the name of the user variable the contains the XPath expression for the XPath filter in the <get-config> operation. The parameter must exist (e.g., created with an assignment statement) or an error message will be printed.
    - **select**
      - type: string
      - usage: mandatory
      - default: none
      - This parameter specifies the XPath expression to use for the XPath filter in the <get-config> operation.
- **source**
  - type: container with 1 of N choice of leafs
  - usage: mandatory
  - default: none
  - This parameter specifies the name of the source database for the retrieval operation.
  - container contents: 1 of N:

        - **candidate**
            - type: empty
            - capabilities needed: :candidate
        - **running**
            - type: empty
            - capabilities needed: none
        - **startup**
            - type: empty
            - capabilities needed: startup
        - **url**
            - type: yang:uri
            - capabilities needed: :url, and the scheme used in the URL must be specified in the :url capability 'schemes' parameter
            - To enter this parameter, the interactive mode must be used. The shorthand mode (source=url) cannot be used, since this parameter contains a string.
- **timeout**
    - type: uint32 (0 = no timeout, otherwise the number of seconds to wait)
    - usage: optional
    - default: set to the **$$timeout** system variable, default 30 seconds
    - This parameter specifies the number of seconds to wait for a response from the agent before giving up. The session will not be dropped if a remote command times out, but any late response will be dropped. A value of zero means no timeout should be used, and **yangcli** will wait forever for a response.
- **with-defaults**
    - type: enumeration (none report-all trim explicit)
    - usage: optional
    - default: none
    - capabilities needed: with-defaults
    - This parameter controls how nodes containing only default values are returned in the <rpc-reply>.

System Variables:

- **$$timeout**
    - The response timeout interval will be derived from this system variable.
- **$$with-defaults**
    - The <with-defaults> parameter for the <get-config> operation will be derived from this system variable.

Positive Response:

- the agent returns <data>

Negative Response:

- An error message will be printed if errors are detected locally.
- An <rpc-error> message will be printed if the agent detected an error.

Output:

- **data**
  - type: anyxml
  - This element will contain the requested data from the **source** database.

Usage:

```
yangcli andy@myagent> xget /nacm/groups \
     source=running

RPC Data Reply 41 for session 10:

rpc-reply {
   data {
      nacm {
         groups {
            group nacm:admin {
               groupIdentity nacm:admin
               userName andy
               userName fred
               userName barney
            }
            group nacm:guest {
               groupIdentity nacm:guest
               userName guest
            }
         }
      }
   }
}

yangcli andy@svnserver>
```

8 netconfd User Guide

9 netconf-central User Guide

10 CLI Reference

Protocol debugging and trace messages can be enabled if the log level is set to 'debug2' or higher.

Note: Log level 'debug3' is very verbose and has an impact on resources used. This log level will cause a lot of YANG debugging messages to be printed.

If a local command encounters an error, such as an invalid parameter or a non-existent file, then 1 or more error messages will be printed. Usually there is a detailed error message, followed by a final status message for the attempted command. Sometimes only the final status message is printed.