Yuma User Manual

YANG-Based Unified Modular Automation Tools

Common User Manual

Version 2.0

Last Updated July 20, 2011

Table Of Contents

Yuma User Manual

1 Preface

1.1 Legal Statements

1.2 Additional Resources

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

Yuma Installation Guide

Other documentation includes:

Yuma Quickstart Guide

Yuma netconfd Manual

Yuma yangcli Manual

Yuma yangdiff Manual

Yuma yangdump Manual

Yuma Developer Manual

To obtain additional support you may join the yuma-users group on sourceforge.net and send email to this e-mail address:

yuma-users@lists.sourceforge.net

The SourceForge.net Support Page for Yuma can be found at this WEB page:

http://sourceforge.net/projects/yuma/support

There are several sources of free information and tools for use with YANG and/or NETCONF.

The following section lists the resources available at this time.

1.2.1 WEB Sites

- **Netconf Central**
  - http://www.netconfcentral.org/
  - Yuma Home Page
    - Free information on NETCONF and YANG, tutorials, on-line YANG module validation and documentation database
- **Yuma SourceFource OpenSource Project**
  - http://sourceforge.net/projects/yuma/

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

1.2.2 Mailing Lists

- **NETCONF Working Group**
    - http://www.ietf.org/html.charters/netconf-charter.html
    - Technical issues related to the NETCONF protocol are discussed on the NETCONF WG mailing list. Refer to the instructions on the WEB page for joining the mailing list.
- **NETMOD Working Group**
    - http://www.ietf.org/html.charters/netmod-charter.html
    - Technical issues related to the YANG language and YANG data types are discussed on the NETMOD WG mailing list. Refer to the instructions on the WEB page for joining the mailing list.

1.3 Conventions Used in this Document

The following formatting conventions are used throughout this document:

Documentation Conventions

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

2 Summary

2.1 What is Yuma?

Yuma 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)
- Partial Lock RPC for NETCONF (RFC 5717)
- YANG Data Modeling Language (RFC 6020)
- Common YANG Data Types (RFC 6021)
- NETCONF Monitoring Schema (RFC 6022)
- With-defaults capability for NETCONF (RFC TBD)
- SSH2 (RFC 4252 - 4254)
- XML 1.0
- XPath 1.0
- YANG Data modeling language (RFC 6020)

The following programs are included in the Yuma 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.

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

- **yuma-netconf.yang**: all the NETCONF protocol operations, including all YANG extensions to the NETCONF protocol (RFC 4741). This file contains meta-data used in the yangcli and netconfd programs, which is not available in the **ietf-netconf.yang** version.
- **ietf-yang-types.yang**: the standard derived types library in progress for YANG. This module is being developed by the NETMOD WG. (RFC 6021)

- **ietf-inet-types.yang**: the standard Internet address types library in progress for YANG. This module is being developed by the NETMOD WG. (RFC 6021)
- **ietf-netconf-monitoring.yang:** the standard NETCONF monitoring module in progress by the NETCONF WG (RFC 6022)
- **ietf-netconf-partial-lock.yang**: the standard NETCONF module for multiple concurrent partial database locks (RFC 5717).
- **ietf-with-defaults.yang**: the standard NETCONF default value control module in progress by the NETCONF WG (draft-ietf-netconf-with-defaults-10.txt)
- **yuma-interfaces.yang**: interfaces monitoring and configuration scaffolding.
- **yuma-mysession.yang**: NETCONF session customization operations
- **notifications.yang**: the standard NETCONF create-subscription command to start receiving NETCONF notifications (RFC 5277)
- **nc-notifications.yang**: the standard NETCONF notifications (RFC 5277)
- **yuma-proc.yang**: /proc file system monitoring information
- **yuma-system.yang**: Proprietary system group and common notifications
- **yuma-nacm.yang**: Proprietary NETCONF Access Control Model
- **test/pass/*.yang**: Several modules are included for testing YANG and NETCONF behavior.
- **test/fail/*.yang**: Several modules with errors are included for testing YANG compiler behavior

2.2 Intended Audience

This document is intended for users of the programs in the Yuma suite.

It contains the following information:

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

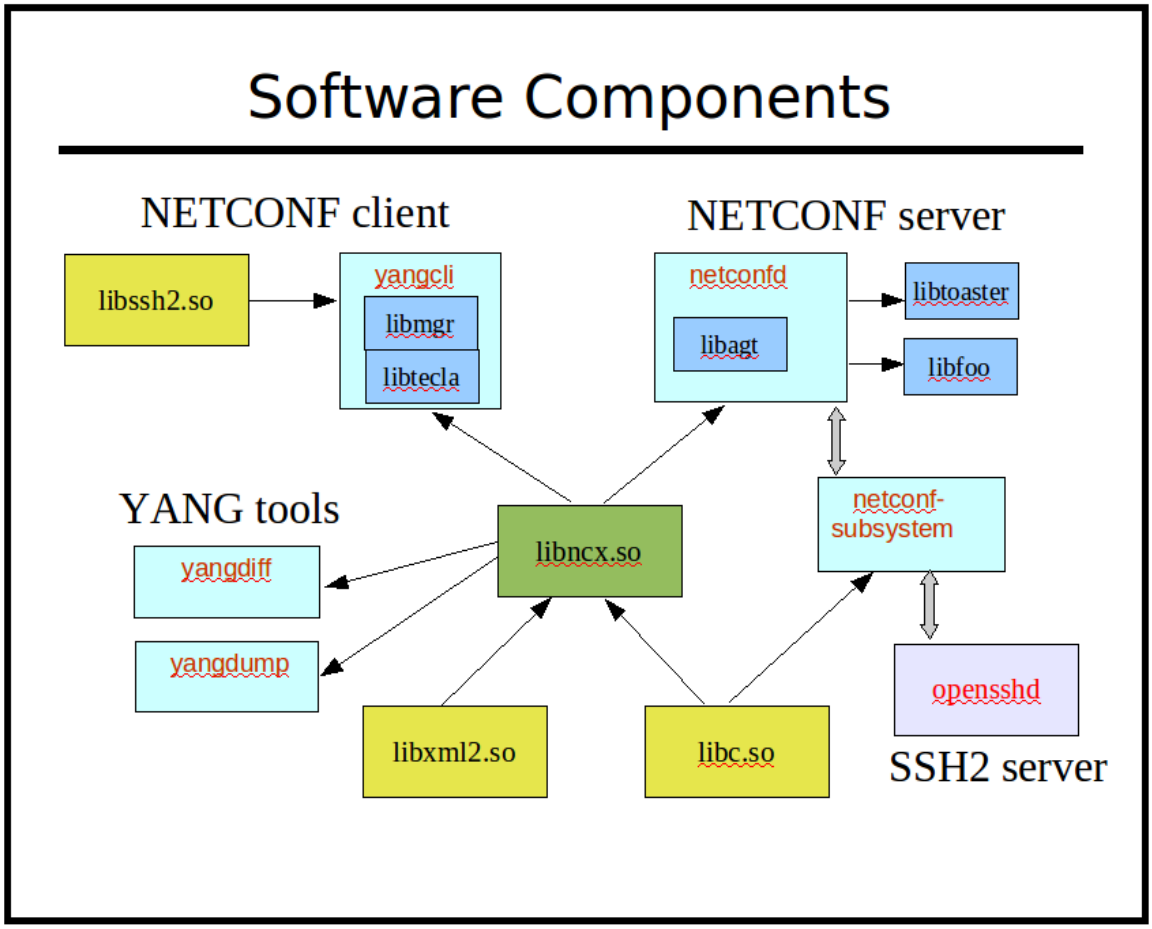
3 Introduction

The Yuma Tools 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



The following external program is used by Yuma, and needs to be pre-installed:

- **opensshd**
  - The SSH2 server code does not link with Yuma. Instead, the **netconf-subsystem** program is invoked, and local connections are made to the **netconfd** server from this SSH2 subsystem.

The following external libraries are used by Yuma, and need to be pre-installed. They are usually installed by default and do not need to be installed by you:

- **libc6**
  - unix system library
- **ncurses**
  - Curses terminal support (needed on Fedora platforms only)
- **libxml2**
  - xmlTextReader XML parser
  - pattern support

The following external library is built within Yuma and does not need to be pre-installed:

- **libtecla**
  - command line support for **yangcli**

The following shared (or static) library is built by Yuma and used by almost all of its programs:

- **libncx**
  - YANG parser
  - YANG validation
  - basic NETCONF support
  - XPath support
  - configuration database support

The following libraries are built by Yuma, and used within executables:

- **libagt**
  - NETCONF server support
- **libmgr**
  - NETCONF client support
- **libydump**
  - yangdump translation functionality

The following binaries are built by Yuma:

- **netconfd**
  - NETCONF server
- **netconf-subsystem**
  - thin client between opensshd and NETCONF server
- **yangcli**
  - NETCONF client
- **yangdump**
  - YANG validation
- **yangdiff**
  - YANG compare

The following sample netconfd module instrumentation library is provided as an example. These libraries (e.g., libfoo.so) can only be created with the Yuma SDK. Refer to the Yuma Developer's Guide for details on creating server instrumentation libraries.

- **libtoaster**
  - Server instrumentation code for the YANG module libtoaster.yang.

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:

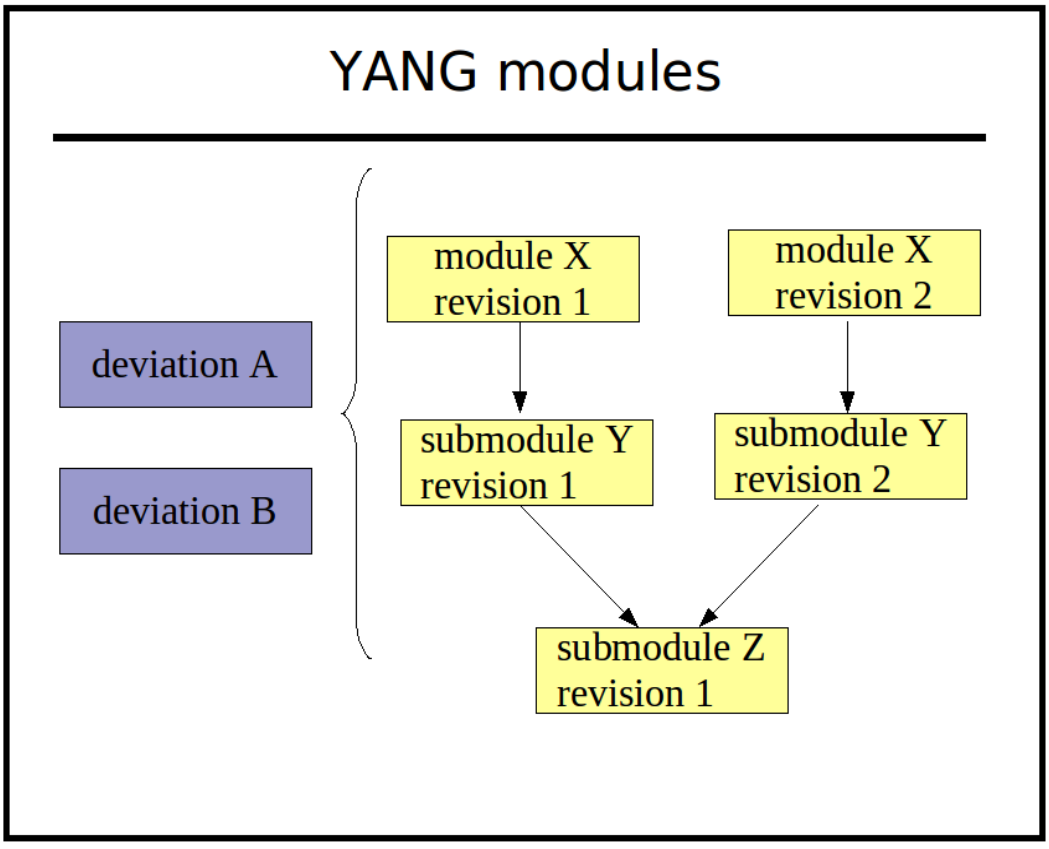
- reusable derived data types
- reusable groupings of objects
- RPC operations
- database objects
- notifications

Network management software developers creating a new management 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 server 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 Yuma to automate many aspects of network management software development.

Semant



A YANG module can be a single file, or it can be split into an arbitrary number of files, using sub-modules. A YANG submodule is essentially the same as a main module, except that the namespace URI value is shared between the main module and all its submodules.

A submodule is referenced with the include statement instead of the import statement.

Submodules can also include other submodules, except a loop may not be formed by the include statements.

Conceptually, the module is not nested. All definitions in submodules appear at the top level of the YANG module, even submodules included by other submodules.

All YANG modules and submodules have revision dates. The example shows a simple version number, but the actual revision strings are date strings in the form 'YYYY-MM-DD'.

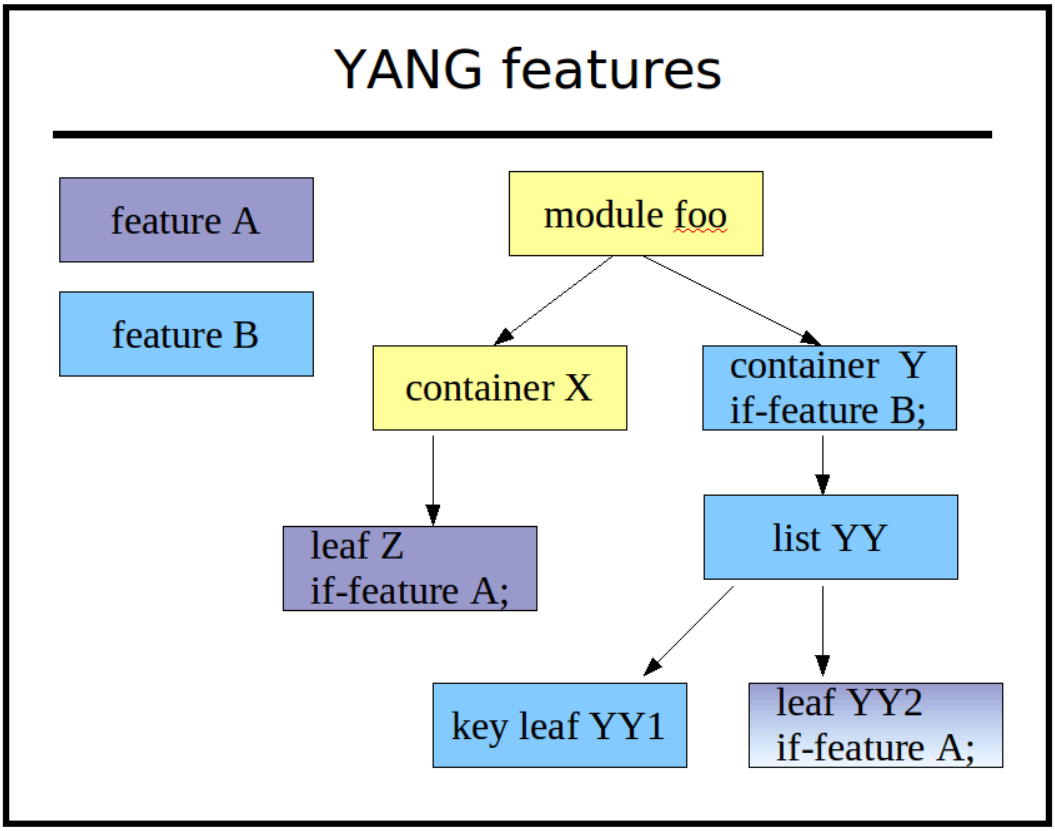
Yuma programs support concurrent usage of different revisions of the same module or submodule. This can occur via groupings from external modules within the YANG language. Only one revision of a module can be imported into a single module or submodule, but any of these files may in turn import other modules. It is possible that a different version of the same module could be indirectly imported in this case.

Deviation modules are normal YANG modules, except they only contain deviation statements. These deviation statements are used to alter (patch) the YANG modules with implementation-specific differences.

A deviation module can contain any number of deviation statements, and they can apply to an arbitrary number of objects, from any module. Multiple deviation statements for the same target will be combined by the server before using them, and all deviate statements for the same object will be validated together, as if they were all contained in the same deviation statement. The order of the deviation statements is irrelevant.

Deviations modules are processed first, and the deviation statements save for later. The import statements are ignored, unlike real module processing.

Since deviation modules are not identified in any way, Yuma programs use the **--module** parameter to refer to a normal YANG module or submodule, and the **--deviation** parameter to refer to a deviation module.



The YANG feature statement is used to define a conceptual partition within the module.

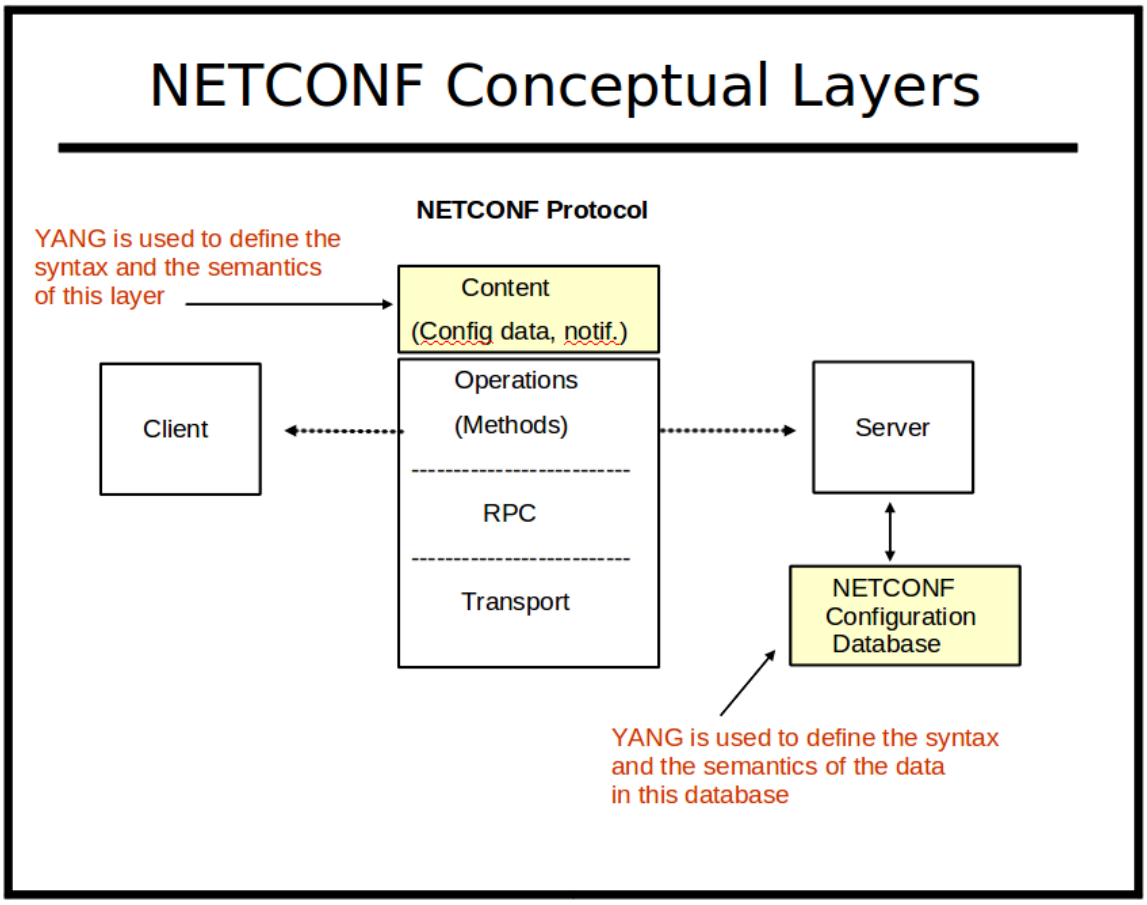
Objects that contain the if-feature statement for the corresponding feature are part of the feature.

If the server does not advertise a feature in its <capabilities>, then it is not supported, and all the objects that are part of the feature are not supported.

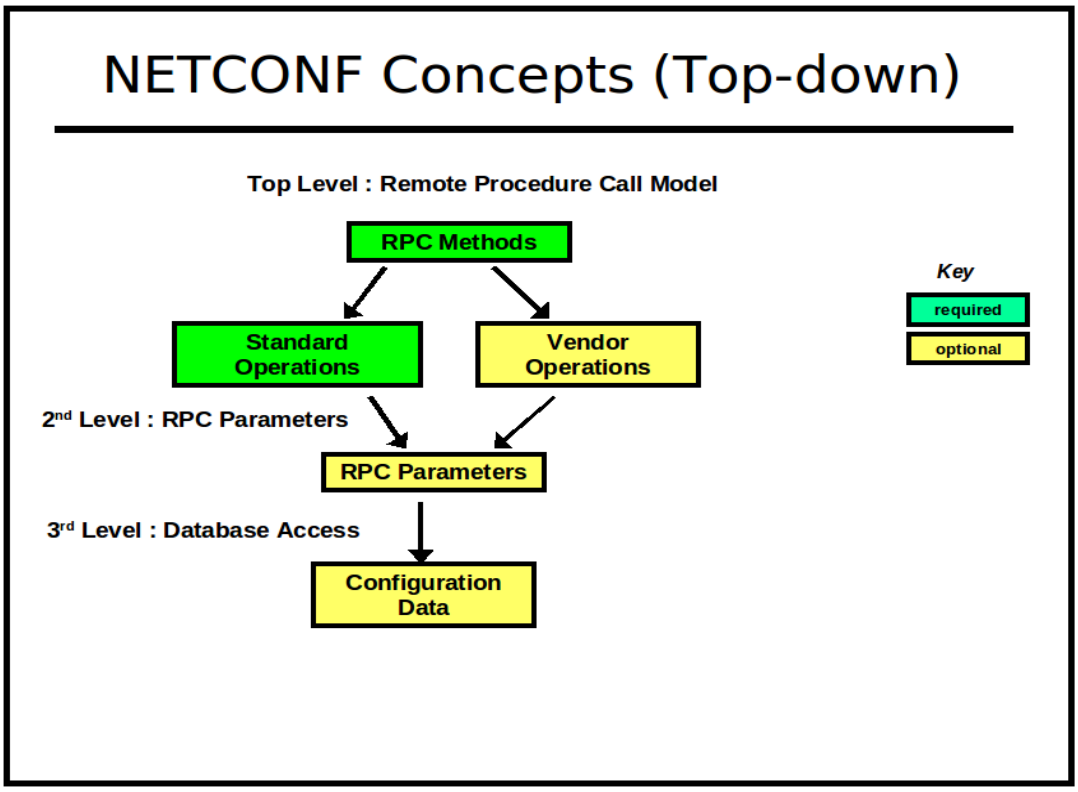
Multiple if-feature statements form a logical AND expression. All the referenced features must be enabled for the object to be available. In the example above, leaf 'YY2' is not present unless feature A and B are both advertised by the server.

3.1.2 NETCONF

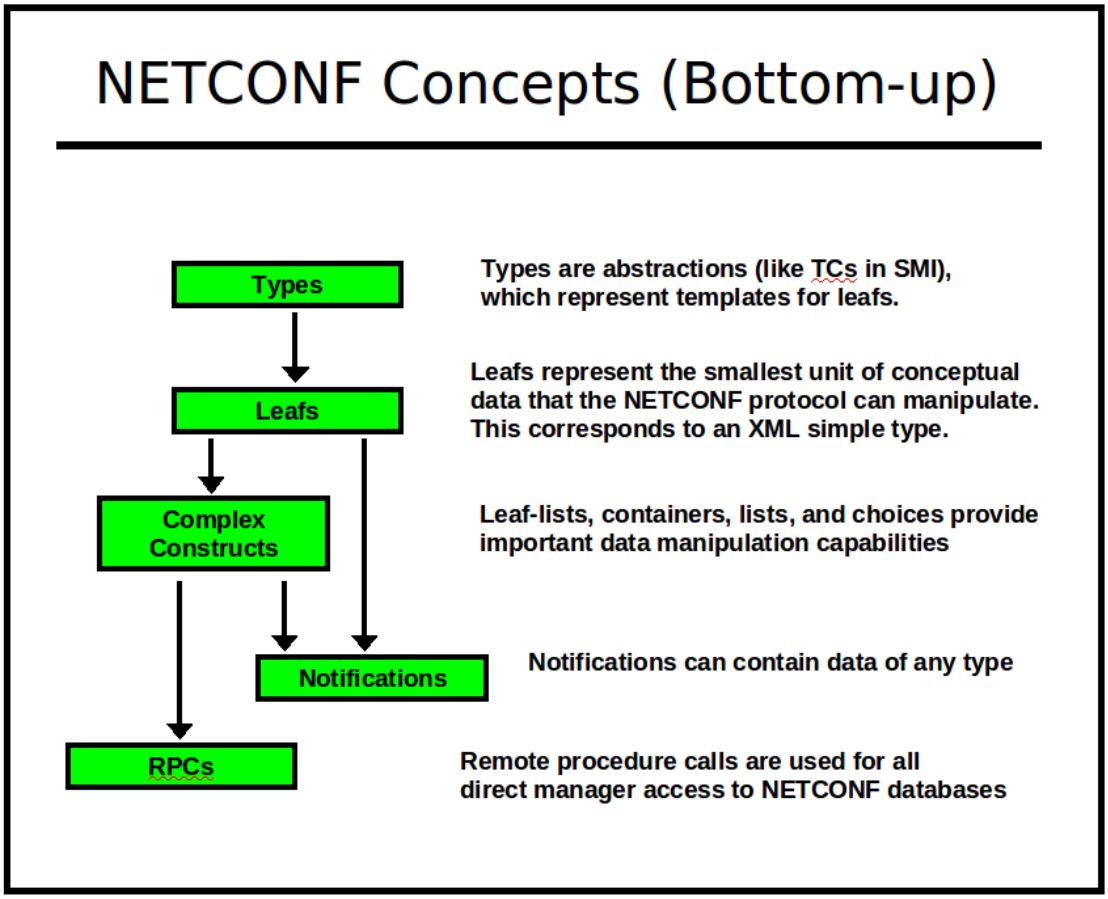
The mandatory components of the NETCONF protocol are defined in RFC 4741 and RFC 4742.



The NETCONF protocol is used to provide secure access all YANG content. The server 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 Yuma. 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.

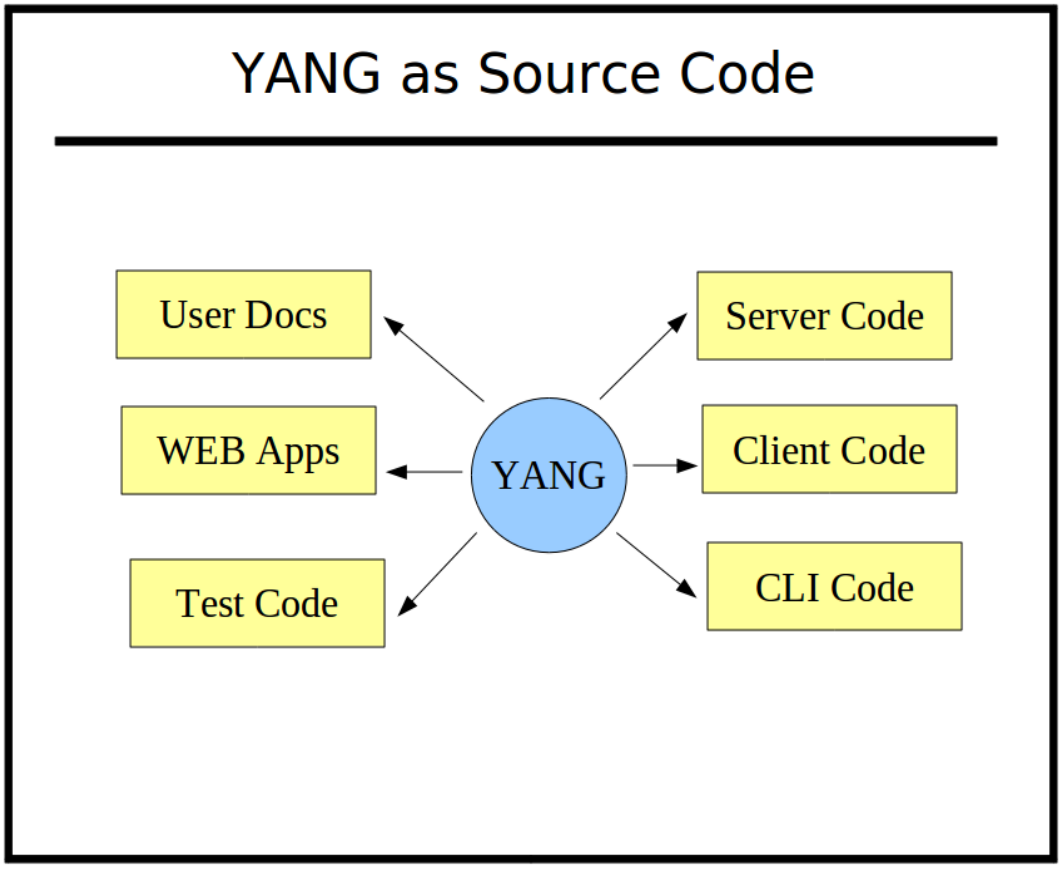
Access control is configurable (via **yuma-nacm.yang**), based on group membership. The access control rules permit or deny access to one or more groups, to a subset of the YANG content. Separate defaults for read, write, and exec (RPC operation) access are provided.

3.1.3 YANG-BASED AUTOMATION

Yuma 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.

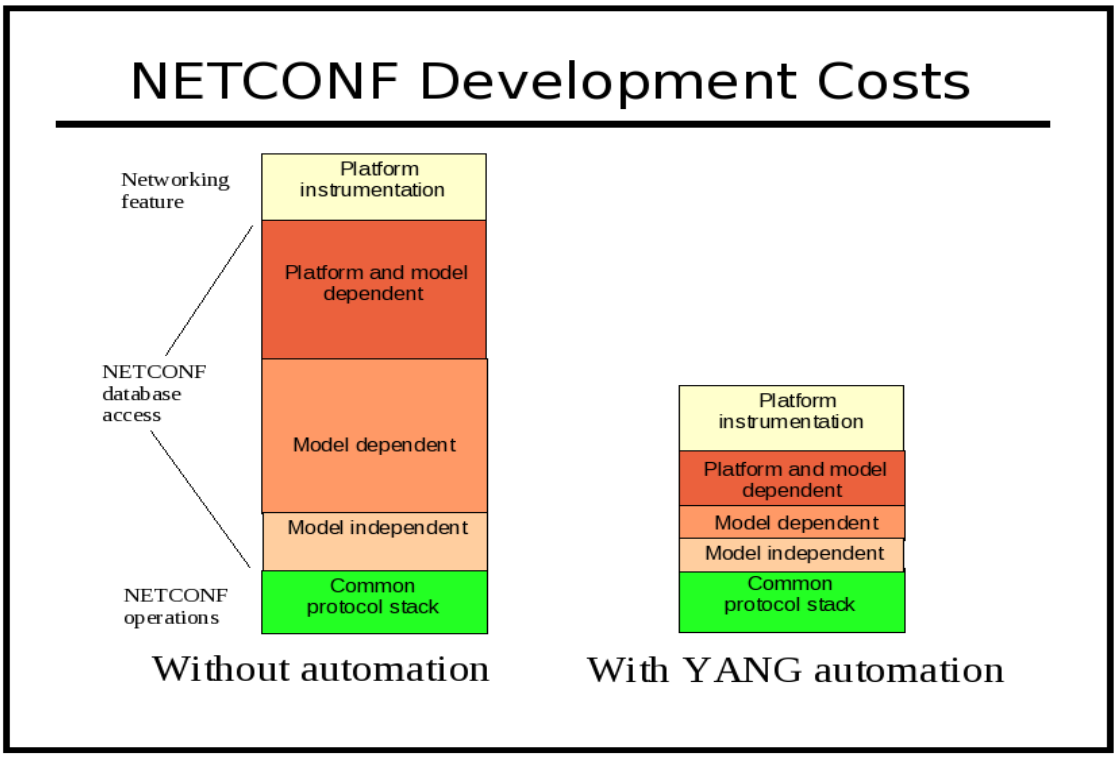
The automation concepts will be familiar to SNMP developers who use SMIv2 to write MIB modules. The SMIv2 language contains enough machine-readable clauses so a client and server can automate certain aspects of the SNMP protocol implmentation.

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** client and **netconfd** server 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
config *boolean*;	The **config** statement indicates if the object is writable, or read-only. The server uses this information when automatically skipping config=false entries for the <get-config> operation.
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 client can rely on it. Defaults can be specified in **typedef** or **leaf** statements. If both are defined, then the leaf default will be used.
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.
error-app-tag *apptag-string*;	The **error-app-tag** statement can be used within the **range**, **length**, and **pattern** statements. If a value is invalid due to the corresponding error, then the <error-app-tag> field in the <rpc-error> sent by the server will be set to the 'apptag-string' value.
error-message *errmsg-string*;	The **error-message** statement can be used within the **range**, **length**, and **pattern** statements. If a value is invalid due to the corresponding error, then the <error-message> field in the <rpc-error> sent by the server will be set to the 'errmsg-string' value.
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.
feature	The **feature** statement allows a module to be conceptually partitioned into mandatory and conditional object groups. All objects with the corresponding if-feature statement will be present only if the feature is supported by the server.
if-feature *feature-name;*	Construct containing the **if-feature** statement is only included if the specified feature is supported by the server. Otherwise, the object does not exist on the server.
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.
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.
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.
mandatory *boolean*;	The **mandatory** statement indicates that the choice, list or leaf must be provided by the client. It will not be created by the server. Most parameters are not mandatory however, so the default is 'false' if this

	statement is missing.
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.
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.
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.
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.
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.
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.
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.
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.
uses *grouping-name*;	The **uses** statement inserts an instance of a reusable **grouping**, replacing the uses node within the conceptual data tree.
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.

3.1.4 YANG Language Extensions

There are several YANG extensions that are supported by Yuma. They are all defined in the YANG file named **yuma-ncx.yang**. They are used to 'tag' YANG definitions for some sort of automatic processing by Yuma 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** server 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 Yuma, 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 server.
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** <data-rule> expressions.

nacm: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.
nacm: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 Yuma 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-13.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 Yuma programs, including netconfd server instrumentation. Refer to the 'YANG User Guide' for details on writing YANG files and using the extensions built into Yuma.

Note: The **smidump** is not part of Yuma, 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.

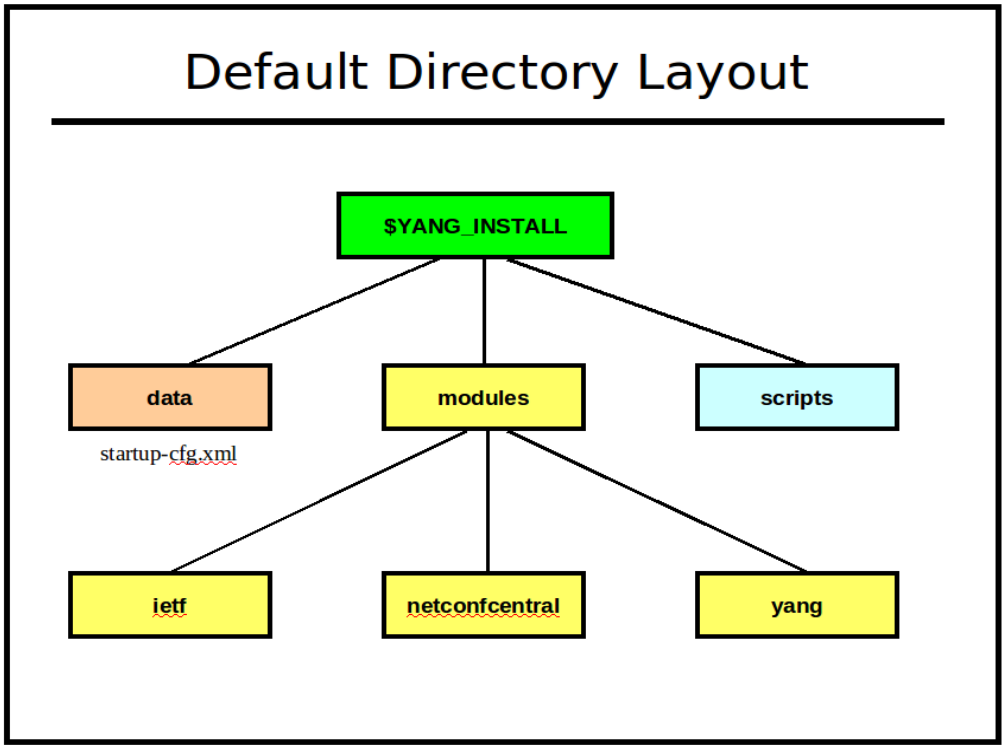
Directory Layout

Yuma 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 (YUMA_INSTALL) shared by all users, and one or more 'project' roots (YUMA_HOME), which can be shared but may belong to a single user.

The Yuma 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.

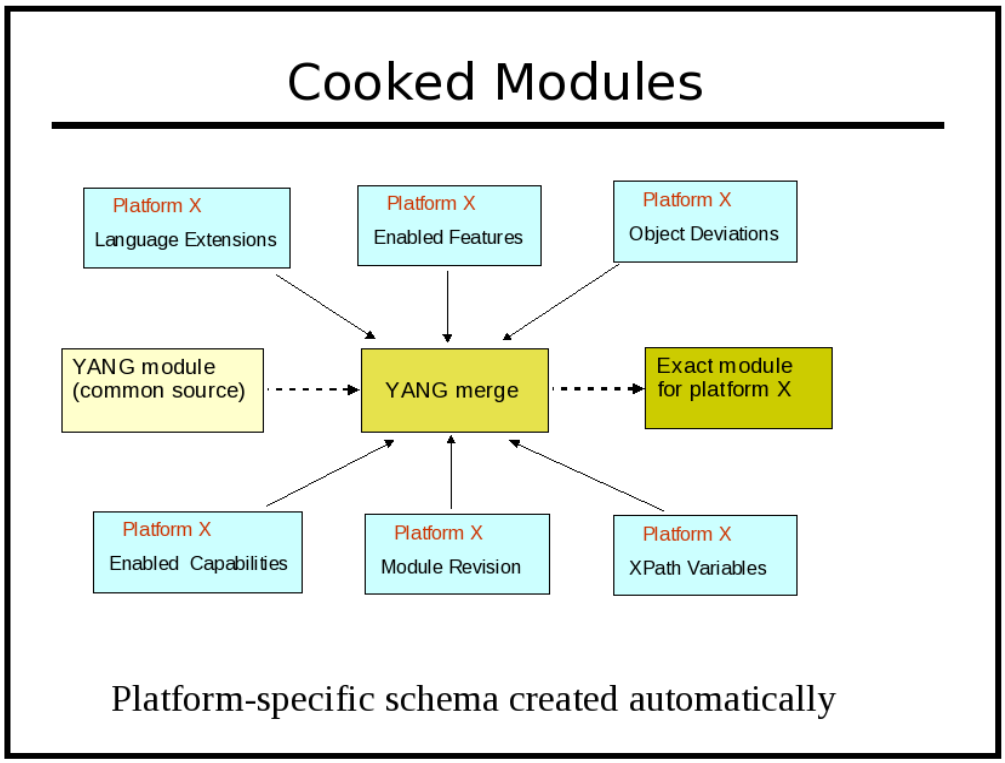


Module Revisions

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 server.

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.

Yuma 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 at sign (@) character be used as a separator between module name string components. YANG files must use the suffix '.yang'. YIN files must use the suffix 'yin'.

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

module.yang

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

module@revision-date.yang

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

These naming conventions are important when Yuma needs to resolve an 'import' or 'include' statement in a YANG file. Refer to section 4.2 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. . There is also an alternate XML encoding called YIN. Sometimes the term YANG module is used to refer to the conceptual module, whether it is encoded in YANG format or YIN format.

All Yuma Tools programs will accept either encoding format, however line and column numbers are not correct in log messages for YIN encoded modules. Instead, each XML node is given a monotonically increasing value, and the XML document order is used instead of line numbers in error/warning messages for YIN files. The column number is always '1' for YIN files.

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 the Developer Manual.
- **server instrumentation code-stubs**: the instrumentation callback functions, used in **netconfd** for activating specific YANG content, can be generated. This procedure is described in more detail in the Developer Manual.
- **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 yuma-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 Managers

The NETCONF client is an application that initiates and utilizes NETCONF sessions to control and monitor a NETCONF server.

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

3.1.9 NETCONF Agents

The NETCONF server is a server application that is always running on the managed device. It listens for NETCONF session requests from a NETCONF client, 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 client, and insulates all the instrumentation code from these protocol operations.

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

4 System Configuration

The Yuma 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 yuma-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 Yuma:

- **yuma-types.yang**: contains common data types used in the Yuma applications
- **yuma-app-common.yang**: contains common CLI parameters used in all Yuma applications
- **yuma-ncx.yang:** contains YANG extensions used in any YANG module, including Yuma 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.

Yuma 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 Yuma 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 Yuma:

- HOME
- YUMA_HOME
- YUMA_INSTALL
- YUMA_MODPATH
- YUMA_DATAPATH
- YUMA_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 Yuma 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:

```
export HOME=/home/andy
```

4.1.2 $YUMA_HOME

The **$YUMA_HOME** environment variable contains the directory specification of the current Yuma project root directory. This is the path to the 'netconf' directory, within a Yuma source tree.

Default value: none

CLI override: --yuma-home

CLI example:

```
--yuma-home=/home/andy/swdev/yuma/trunk/netconf
```

C shell example:

```
setenv YUMA_HOME  /home/andy/swdev/yuma/trunk/netconf
```

Bash shell example:

```
export YUMA_HOME=/home/andy/swdev/yuma/trunk/netconf
```

4.1.3 $YUMA_INSTALL

The **$YUMA_INSTALL** environment variable contains the directory specification of the Yuma installation root directory.

Default value: /usr/share/yuma

CLI override: none

C shell example:

```
setenv YUMA_INSTALL  /sw/yuma
```

Bash shell example:

```
export YUMA_INSTALL=/sw/yuma
```

4.1.4 $YUMA_MODPATH

The **$YUMA_MODPATH** environment variable contains a list of directory specifications that should be searched (in order) to find YANG or YIN 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 **--subdirs** parameter. Refer to the Command Line Parameter Reference for more details. If **--subdirs=false** 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 **$HOME**, **$YUMA_HOME**, and **$YUMA_INSTALL** environment variables, which specify a Yuma root directory.

Default value: none

CLI override: --modpath

CLI example:

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

C shell example:

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

Bash shell example:

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

4.1.5 $YUMA_DATAPATH

The **$YUMA_DATAPATH** environment variable contains a list of directory specifications that should be searched (in order) to find data files used by Yuma 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 server 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 **--subdirs** parameter. Refer to the Command Line Parameter Reference for more details. If **--subdirs=false** 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 **$HOME**, **$YUMA_HOME**, and **$YUMA_INSTALL** environment variables, which specify a Yuma root directory.

Default value: none

CLI override: --datapath

CLI example:

```
--datapath=”$HOME/mydata:$HOME/project1/data”
```

C shell example:

```
setenv YUMA_DATAPATH  “$HOME/mydata:$HOME/project1/data”
```

Bash shell example:

```
export YUMA_DATAPATH="$HOME/mydata:$HOME/project1/data"
```

4.1.6 $YUMA_RUNPATH

The **$YUMA_RUNPATH** environment variable contains a list of directory specifications that should be searched (in order) to find script files used by Yuma 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 **--subdirs** parameter. Refer to the Command Line Parameter Reference for more details. If **--subdirs=false** 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 **$HOME**, **$YUMA_HOME**, and **$YUMA_INSTALL** environment variables, which specify a Yuma root directory.

Default value: none

CLI override: --runpath

CLI example:

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

C shell example:

```
setenv YUMA_RUNPATH  "$HOME/scripts:/usr/local/scripts"
```

Bash shell example:

```
export YUMA_RUNPATH="$HOME/scripts:/usr/local/scripts"
```

4.2 Searching for Files

All Yuma programs search for YANG and other 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 Yuma 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
$YUMA_HOME/data/startup-cfg.xml
```

Note: Whenever Yuma 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
- $YUMA_HOME
- $YUMA_MODPATH
- $YUMA_DATAPATH
- $YUMA_RUNPATH

The following configuration parameters affect file searches:

- --yuma-home
- --modpath
- --datapath
- --runpath
- --subdirs

4.2.1 Yuma Work Directory

There is a directory (**$HOME/.yuma**) created by **yangcli** or **netconfd** for data files and temporary files. It is called **.yuma**, and it is created in the users home directory, if the **$HOME** environment variable is defined.

This directory will be used as the default location to save the **startup-cfg.xml** file by **netconfd**, if no startup file is specified in the CLI parameters, and no existing startup file is found in the data file search path.

This directory is also used as the default location to store the **.yangcli_history** file for **yangcli** command line history recall.

The **$HOME/.yuma/tmp** directory is used by **yangcli** to create session-specific sub-directories where all the YANG modules from the server for the current session are stored. If the **--autoload=false** parameter is used, then these temporary directories will not be created by **yangcli**.

4.2.2 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.

4.2.3 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.

4.2.4 File Search Paths

Yuma 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 **$YUMA_MODPATH** environment variable, or set with the **–modpath** configuration parameter, is checked.
   - If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
3. The **$HOME/modules** directory is checked.
   - If the **--subdirs=false** 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 **$YUMA_HOME/modules** directory is checked.
   - If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
5. The **$YUMA_INSTALL/modules** directory is checked.
   - If the **--subdirs=fasle** 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

Yuma 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. Each directory specified in the **$YUMA_DATAPATH** environment variable, or set with the **–datapath** configuration parameter, is checked.
   1. If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
3. The current working directory is checked. No sub-directories are checked, if any are present
4. The **$HOME/data** directory is checked.
   1. If the **--subdirs=false** 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 **$YUMA_HOME/data** directory is checked.
   1. If the **--subdirs=false** 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 **$HOME/.yuma** directory is checked.
7. The **$YUMA_INSTALL/data** directory is checked.
    1. If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
8. The **/usr/share/yuma/data** directory is checked.
    1. If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
9. The **/etc/yuma** directory is checked.

Script Search Path

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

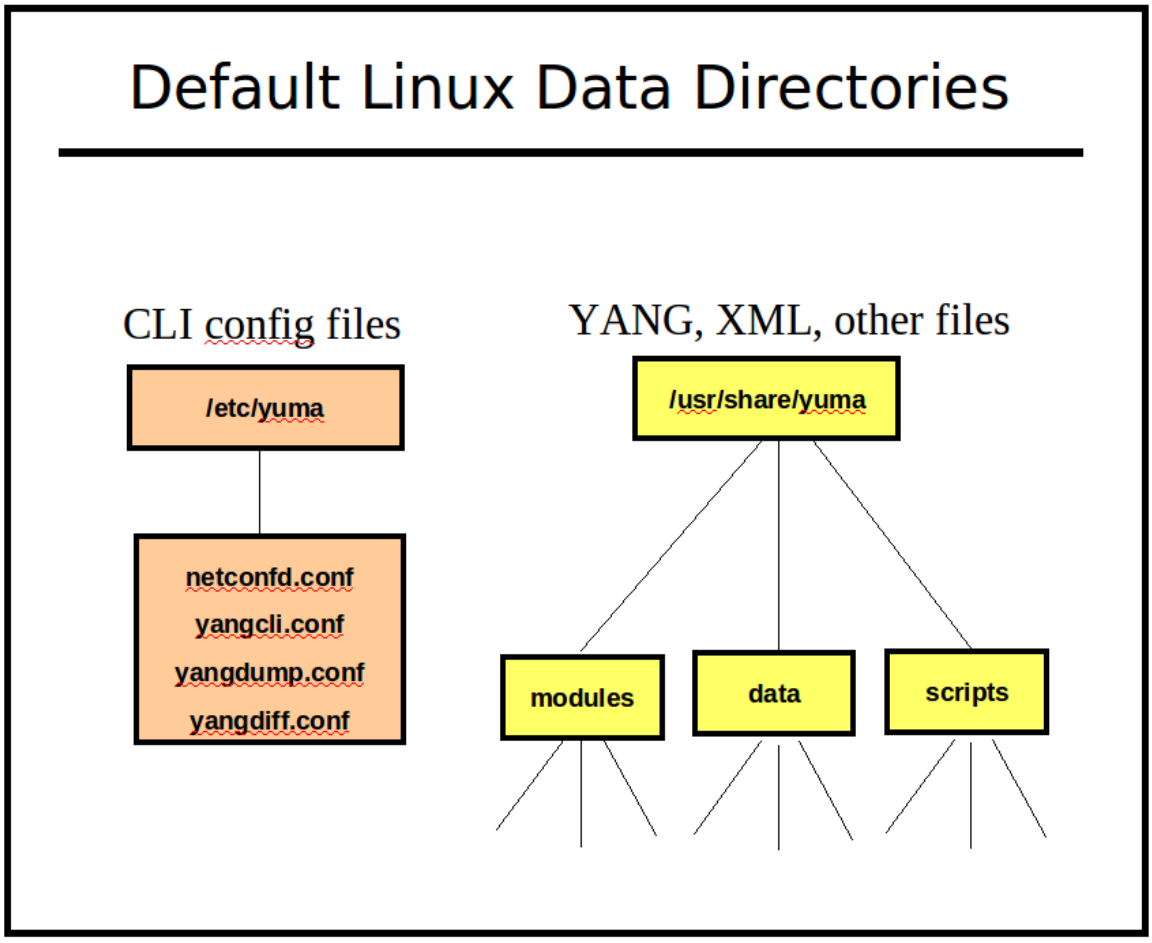
1. If the file name has an absolute path specification, then that exact file location is tried. If no match is found, then the search will terminate in failure.
2. The current working directory is checked. No sub-directories are checked, if any are present.
3. Each directory specified in the **$YUMA_RUNPATH** environment variable, or set with the **--runpath** configuration parameter, is checked.
    - If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
2. The **$HOME/scripts** directory is checked.
    - If the **--subdirs=false** 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 **$YUMA_HOME/scripts** directory is checked.
    - If the **--subdirs=false** parameter is set, then only each top-level directory will be checked. If it is not set, then sub-directories will be searched.
4. The **$YUMA_INSTALL/scripts** directory is checked.
    - If the **--subdirs=false** 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 Yuma 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
- $YUMA_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 (yuma-nacm.yang), in XML configuration file format.

```
// file startup-cfg.xml
<?xml version="1.0" encoding="UTF-8"?>
<nd:config xmlns:nd="http://netconfcentral.org/ns/netconfd">
   <nacm:nacm xmlns:nacm="http://netconfcentral.org/ns/yuma-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:allowed-rights>read write exec</nacm:allowed-
rights>
            <nacm:allowed-group>nacm:admin</nacm:allowed-group>
         </nacm:moduleRule>
      </nacm:rules>
   </nacm:nacm>
</nd:config>
```

4.3.2 Text Configuration Files

The Yuma 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 following configuration parameters affect generation and display of text configuration files
  - --indent
  - --with-defaults
  - --display-mode

4.4 Bootstrap CLI

Since Yuma 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 **$YUMA_MODPATH** environment variable, if it is set
- **--yuma-home:** use the specified project root. This will override the **$YUMA_HOME** environment variable, if it is set

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

4.5 Configuration Parameters

Command line parameters are used to provide input to Yuma 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 **yuma-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 **yuma-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 XPath Reference

The XPath 1.0 path specification language is supported in all Yuma Tools programs, as specified in the YANG language specification. There are also some additional variables and XPath functions, available in all XPath expressions.

A custom XPath implementation is used, which is based on the internal data structures maintained within the program (i.e., object tree or data tree). No CPU or memory is wasted converting these data structures to actual XML documents for XPath processing.

5.1 XPath 1.0

All functionality defined in the XPath 1.0 specification is supported.

There are some restrictions, which are specific to the YANG standard:

- The 'attribute' and 'processing-instruction' axes are always empty.
- YANG identityref leaf values need to be entered within quotes or they will be interpreted as XML qualified node names.
- The server may not maintain consistent XML document order for system-ordered data. This affects expressions which rely on XML document order to be precise and completely static. A NETCONF server is only required to maintain XML document order for user-ordered lists and leaf-lists, and only relative to a particular object, not the entire document.

5.1.1 XML Namespaces

The XPath implementation allows a more liberal syntax than the XPath 1.0 specification allows.

Specifically, if a node identifier does is unqualified (i.e., there is no namespace specified with a default namespace or an explicit namespace declaration), then all known XML namespaces known by the program will be checked for a top-level element with the same name.

- **If XML namespaces are used, they must be used correctly.**

Example request using XML namespaces in an XPath expression:

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
   message-id="2"
   xmlns:sys="http://netconfcentral.org/ns/yuma-system">
   <nc:get>
      <nc:filter type="xpath" select="/sys:system"/>
   </nc:get>
</nc:rpc>
```

Note the text:

```
xmlns:sys="http://netconfcentral.org/ns/yuma-system"
```

This 'xmlns' attribute does not have to appear exactly as specified, or within the <rpc> element. It can appear in any legal manner. Refer to the **XML Namespaces 1.0** specification for more details.

Example request not using XML namespaces in an XPath expression:

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
```

```
   message-id="3">
   <nc:get>
      <nc:filter type="xpath" select="/system"/>
   </nc:get>
</nc:rpc>
```

If the 'yuma-system.yang' module is loaded within the program, and if the 'system' node is enabled (e.g., not removed via a YANG deviation), then the XML prefix ('sys:' in this example) can be omitted.

5.2 YANG Specific XPath Behavior

The YANG language requires some minor changes and additions to the XPath 1.0 specification:

- The 'current' function from XPath 2.0 is supported.
- The NULL XML namespace is mapped to the current YANG module XML namespace, when processing an XPath expression within a YANG module (e.g., must statement).
- A NETCONF database is treated as a conceptual XML instance document with zero or more top-level elements. This is consistent with XSLT behavior. XML 1.0 requires a single top-level element,so external XML documents representing a NETCONF database always start with the <nc:config> element (config element in the NETCONF XML namespace).

5.3 Custom XPath Variables

The XPath specification supports system variables to be accessed within XPath expressions.

Within the **yangcli** program, all user and system variables available to a script are also available as XPath variables within XPath expression evaluation (e.g., if, eval, and while commands).

For example, a variable named 'myvar' would be accessed within an XPath expression as '$myvar'.

5.3.1 USER

An XPath variable called 'user' is supported in the **yangcli** and **netconfd** programs. It is equal to the NETCONF user name associated with the session evaluating the XPath expression. It is provided to be used in data rules within the NETCONF Access Control Model (NACM).

5.4 Custom XPath Functions

The following XPath functions are added to the XPath 1.0 Function Library, in addition to the 'current' function from XPath 2.0.

5.4.1 MODULE-LOADED

The **module-loaded** function tests whether the specified module is loaded within the program.

boolean module-loaded (module-name [, revision-date])

Parameters:

- Parameter 1:
  - Type: String
  - Usage: Mandatory

    - Purpose: Specifies the module name to check.
- Parameter 2:
    - Type: String
    - Usage: Optional
    - Purpose: Specifies the YANG revision date string for module indicated by parameter 1.

Returns: Boolean

- true: the specified module is loaded
- false: the specified module is not loaded, possibly not known

Errors:

- Missing parameter error if no parameters provided.
- Extra parameters error if more than 2 parameters provided.
- All unknown parameter values cause a 'false' result.

Example:

```
yangcli> if "module-loaded('yuma-system', '2009-12-27')"
yangcli>    log-info 'correct yuma-system module loaded'
yangcli> else
yangcli>    log-error 'Wrong yuma-system module loaded'
yangcli> end
```

5.4.2 FEATURE-ENABLED

The **feature-enabled** function tests whether the specified YANG feature is enabled within the program.

boolean feature-enabled (module-name, feature-name)

Parameters:

- Parameter 1:
    - Type: String
    - Usage: Mandatory
    - Purpose: Specifies the module name to check.
- Parameter 2:
    - Type: String
    - Usage: Mandatory
    - Purpose: Specifies the YANG feature name defined within the module indicated by parameter 1.

Returns: Boolean

- true: the specified feature is enabled
- false: the specified feature is not enabled, possibly not known

Errors:

- Missing parameter error if less than 2 parameters provided.
- Extra parameters error if more than 2 parameters provided.

- All unknown parameter values cause a 'false' result.

Example:

```
yangcli> if "feature-enabled('mymodule', 'myfeature')"
yangcli>    log-info 'myfeature is enabled'
yangcli> else
yangcli>    log-error 'myfeature is not enabled'
yangcli> end
```

6 Error Reference

All Yuma programs use the same set of error numbers and error messages.

Error numbers are 3 digit unsigned integers in the range 1 to 999. The number 0 is reserved for the NO_ERR constant, which is the same as the <ok/> status returned by the server.

Error Number Types

range	description
100 to 199	system errors
200 to 399	user errors
400 to 899	warnings
900 to 999	informational messages

6.1 Error Messages

The current list of error numbers and default error messages can be obtained with the **yangdump** program **--show-errors** parameter.

The default error message can be replaced for some error conditions with the YANG error-message statement.

The following list shows the default error messages for all error numbers currently in use.

```
yangdump  errors and warnings

  0       ok
  1       EOF reached
  2       NULL pointer
  3       malloc failed
  4       invalid internal value
  5       internal buffering failed
  6       invalid queue deletion
  7       wrong init sequence
  8       queue node not header
  9       queue node not data
 10       invalid queue header
 11       entry already queued
 12       too many entries
 13       libxml2 operation failed
100       cannot open file
101       cannot read file
102       cannot close file
103       cannot write file
104       cannot change directory
105       cannot stat file
106       buffer overflow error
```

```
107     cannot delete file
108     cannot access file
109     db connect failed
110     db entry exists
111     db not found
112     db query failed
113     db delete failed
114     wrong checksum
115     wrong tag type
116     db read failed
117     db write failed
118     db init failed
119     beep init failed
120     beep init nc failed
121     xml reader internal
122     open directory failed
123     read directory failed
200     no config file
201     no source file
202     POST read input
203     bad drive
204     bad path
205     bad filename
206     duplicate value pair
207     page not handled
208     page access denied
209     missing form params
210     invalid form state
211     duplicate namespace
212     xml reader start failed
213     xml reader read failed
214     wrong XML node type
215     xml reader null name
216     xml reader null value
217     xml reader wrong name
218     xml reader wrong value
219     xml reader wrong element
220     xml reader extra nodes
221     xml reader EOF
222     wrong length
223     entry exists
224     duplicate entry
225     not found
226     missing file
227     unknown parameter
228     invalid name
229     unknown namespace
230     wrong namespace
231     wrong data type
232     wrong value
233     missing parameter
234     extra parameter
235     empty value
236     module not found
237     max length exceeded
238     invalid token
239     unended quoted string
240     read failed
241     invalid number
242     invalid hex number
243     invalid real number
244     EOF reached
245     wrong token type
```

```
246     wrong token value
247     buffer overflow
248     invalid range
249     overlapping range
250     definition not found
251     definition segment not found
252     type not allowed in index
253     index type not found
254     type not mdata
255     meta-data not allowed
256     top not found
257     resource in use
258     invalid value
259     too big
260     missing attribute
261     bad attribute
262     unknown attribute
263     missing element
264     bad element
265     unknown element
266     unknown namespace
267     access denied
268     lock denied
269     resource denied
270     rollback failed
271     data exists
272     data missing
273     operation not supported
274     operation failed
275     partial operation
276     wrong namespace
277     wrong node depth
278     wrong owner
279     wrong element
280     wrong order
281     extra node
282     wrong node type
283     expecting complex node type
284     expecting string node type
285     wrong data type
286     wrong data value
287     invalid number length
288     value not in range
289     wrong number type
290     invalid enum value
291     value not in set
292     extra list string found
293     unknown object
294     extra parameter instance
295     extra case in choice
296     missing mandatory choice
297     wrong config state
298     unknown application
299     unknown data type
300     access control violation
301     config locked
302     wrong config state
303     max-access exceeded
304     wrong index type
305     wrong instance type
306     missing index component
307     config not found
308     extra attribute instance(s) found
```

```
309     required attribute not found
310     required value instance not found
311     extra value instance(s) found
312     target is read only
313     invalid pattern
314     wrong version
315     connect failed
316     unknown host
317     session failed
318     authentication failed
319     end of comment not found
320     invalid string concatenation
321     import not found
322     missing typedef sub-section
323     restriction not allowed for this type
324     specified refinement not allowed
325     definition loop detected
326     default case contains mandatory object(s)
327     import loop
328     include loop
329     expecting module
330     expecting submodule
331     undefined prefix
332     imported module has errors
333     pattern match failed
334     invalid data type change
335     mandatory object not allowed
336     unique-stmt test failed
337     max-elements exceeded
338     min-elements not reached
339     must-stmt test failed
340     data restriction violation
341     missing instance for insert operation
342     object not config
343     invalid conditional object
344     using obsolete definition
345     invalid augment target
346     duplicate refine sub-clause
347     invalid deviate sub-clause
348     invalid XPath expression syntax
349     invalid instance-identifier syntax
350     require-instance test failed
351     key or select attribute not allowed
352     invalid unique-stmt node
353     invalid duplicate import-stmt
354     invalid duplicate include-stmt
355     ambiguous command
356     unknown module
357     unknown version
358     value not supported
359     leafref path loop
360     variable not found
361     variable is read-only
362     decimal64 base number overflow
363     decimal64 fraction precision overflow
364     when-stmt tested false
365     no matches found
366     missing refine target
367     candidate cannot be locked, discard-changes needed
368     timeout occurred
369     multiple module revisions exist
370     XPath result not a nodeset
371     XPath node-set result is empty
```

372	node is protected by a partial lock
373	cannot perform the operation with confirmed-commit pending
374	cannot directly load a submodule
375	cannot write to a read-only object
376	cannot write to this configuration directly
377	YANG file missing right brace
378	invalid protocol framing characters received
379	base:1.1 protocol not enabled
380	persistent confirmed commit not active
400	duplicate source
401	include file not found
402	invalid command line value
403	invalid command line option
404	command line option unknown
405	invalid command line syntax
406	missing command line value
407	invalid form input
408	invalid form
409	no instance found
410	session closed by remote peer
411	duplicate import
412	duplicate import with different prefix value
413	local typedef not used
414	local grouping not used
415	import not used
416	duplicate unique-stmt argument
417	statement ignored
418	duplicate include
419	include not used
420	revision date before 1970
421	revision date in the future
422	enum value order
423	bit position order
424	invalid status for child node
425	duplicate sibling node name from external augment
426	duplicate if-feature statement
427	using deprecated definition
428	XPath object predicate check limit reached
429	empty XPath result in must or when expr
430	no ancestor node available
431	no parent node available
432	no child node available
433	no descendant node available
434	no nodes available
435	bad revision-stmt order
436	duplicate prefix
437	identifier length exceeded
438	display line length exceeded
439	received unknown capability
440	invalid module capability URI
441	unknown child node, using anyxml
442	invalid value used for parm
443	changing object type to string
444	using a reserved name
445	conf file parm already exists
446	no valid revision statements found
447	dependency file has errors
448	top-level object is mandatory
449	file name does not match [sub]module name