



# **YumaPro User Manual**

---

YANG-Based Unified Modular Automation Tools

Common User Manual

Version 19.10-12

## Table of Contents

1	Preface.....	4
1.1	Legal Statements.....	4
1.2	Additional Resources.....	4
1.2.1	WEB Sites.....	4
1.2.2	Mailing Lists.....	5
1.3	Conventions Used in this Document.....	5
2	Summary.....	6
2.1	What is YumaPro?.....	6
2.2	Intended Audience.....	8
3	Introduction.....	9
3.1	System Components.....	9
3.1.1	YANG.....	11
3.1.2	NETCONF.....	14
3.1.3	YANG-based Automation.....	17
3.1.4	YANG Language Extensions.....	21
3.1.5	YANG Compiler.....	22
3.1.6	YANG Module Library.....	23
3.1.7	YANG Files.....	26
3.1.8	NETCONF Managers.....	26
3.1.9	NETCONF Servers.....	26
4	System Configuration.....	27
4.1	Environment Variables.....	28
4.1.1	\$HOME.....	28
4.1.2	\$YUMAPRO_HOME.....	29
4.1.3	\$YUMAPRO_INSTALL.....	30
4.1.4	\$YUMAPRO_MODPATH.....	31
4.1.5	\$YUMAPRO_LOADPATH.....	32
4.1.6	\$YUMAPRO_DATAPATH.....	33
4.1.7	\$YUMAPRO_RUNPATH.....	34
4.2	Searching for Files.....	35
4.2.1	YumaPro Work Directory.....	37
4.2.2	Parameter Searches.....	37
4.2.3	Import/Include Searches.....	38
4.2.4	File Search Paths.....	40
4.3	Configuration Files.....	42
4.3.1	XML Configuration Files.....	43
4.3.2	Text Configuration Files.....	44
4.4	Logging.....	46
4.4.1	Logging Output Stream Commands.....	47
4.4.2	Logging Detail Commands.....	47
4.4.3	Logging Debug and Development Commands.....	48
4.4.4	Logging Examples.....	49
4.4.5	Logging Backtrace Information.....	53
4.4.6	Logging Hints.....	54
4.5	Pthreads Images.....	55
4.6	Bootstrap CLI.....	56
4.7	Configuration Parameters.....	57
4.7.1	Parameter Syntax.....	57
4.7.2	ncx:cli Extension.....	58

# YumaPro User Manual

4.7.3 ncx:default-parm Extension.....	58
5 XPath Reference.....	60
5.1 XPath 1.0.....	60
5.1.1 XML Namespaces.....	60
5.2 YANG Specific XPath Behavior.....	62
5.3 Custom XPath Variables.....	62
5.3.1 user.....	62
5.4 Custom XPath Functions.....	63
5.4.1 module-loaded.....	63
5.4.2 feature-enabled.....	64
6 Error Reference.....	65
6.1 Error Messages.....	65

# 1 Preface

## 1.1 Legal Statements

Copyright 2009 - 2012, Andy Bierman, All Rights Reserved.

Copyright 2012 - 2020, YumaWorks, Inc., All Rights Reserved.

## 1.2 Additional Resources

This document assumes you have successfully set up the software as described in one or both of the printed documents:

YumaPro Installation Guide

Other documentation includes:

YumaPro API Quickstart Guide

YumaPro Quickstart Guide

YumaPro netconfd-pro Manual

YumaPro yangcli-pro Manual

YumaPro yangdiff-pro Manual

YumaPro yangdump-pro Manual

YumaPro Developer Manual

YumaPro ypclient-pro Manual

YumaPro yp-system API Guide

YumaPro yp-show API Guide

YumaPro Yocto Linux Quickstart Guide

YumaPro yp-snmp Manual

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

[support@yumaworks.com](mailto:support@yumaworks.com)

### 1.2.1 WEB Sites

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

## YumaPro User Manual

- <http://www.yang-central.org>
- Free information and tutorials on YANG, free YANG tools for download
- **NETCONF Working Group Wiki Page**
  - <http://trac.tools.ietf.org/wg/netconf/trac/wiki>
  - Free information on NETCONF standardization activities and NETCONF implementations
- **NETCONF WG Status Page**
  - <http://tools.ietf.org/wg/netconf/>
  - IETF Internet draft status for NETCONF documents
- **libsmi Home Page**
  - <http://www.ibr.cs.tu-bs.de/projects/libsmi/>
  - Free tools such as smidump, to convert SMIV2 to YANG

### 1.2.2 Mailing Lists

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

### 1.3 Conventions Used in this Document

The following formatting conventions are used throughout this document:

#### Documentation Conventions

Convention	Description
<b>--foo</b>	CLI parameter foo
<b>&lt;foo&gt;</b>	XML parameter foo
<b>foo</b>	<b>yangcli-pro</b> command or parameter
<b>\$FOO</b>	Environment variable FOO
<b>\$\$foo</b>	<b>yangcli-pro</b> global variable foo
some text	Example command or PDU
some text	Plain text

## 2 Summary

### 2.1 What is YumaPro?

YumaPro 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)
- YANG Module Library (RFC 7895)

The following programs are included in the YumaPro suite:

- **yangdump-pro**: validates YANG modules and uses them to generate other formats, such as HTML, XSD, SQL, and C source code
- **yangdiff-pro**: reports semantic differences between two revisions of a YANG module, and generates YANG revision statements
- **yangcli-pro**: NETCONF over SSH client, providing a simple but powerful command line interface for management of any NETCONF content defined in YANG
- **netconfd-pro**: NETCONF over SSH server, providing complete and automated support for the YANG content accessible with the NETCONF protocol
- **netconf-subsystem-pro**: thin client used to allow OpenSSH to communicate with the netconfd-pro program. This is documented as part of the **netconfd-pro** program, since they must be used together.
- **yp-shell**: CLI client used to provide console access to **netconfd-pro** server
- **yang-api**: FastCGI client used to provide HTTP/REST access to **netconfd-pro** server using YANG-API protocol
- **restconf**: FastCGI client used to provide HTTP/REST access to **netconfd-pro** server using RESTCONF protocol
- **ypwatcher**: program that provides monitoring mechanism to **netconfd-pro** server and its state. **Ypwatcher** program periodically checks the server's state and determine if the server is still running. If the server is no longer running it cleans up the state, restarts the server, and generates a syslog message.

## YumaPro User Manual

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

The following content (YANG modules) are optional to the **netconfd-pro** server, and supported by the **yangcli-pro** client and **netconfd-pro** only if specific build variable is set during build procedure:

- **yuma-interfaces.yang**: interfaces monitoring and configuration scaffolding. Will be available if WITH\_YUMA\_INTERFACES=1 build variable is set.
- **yuma-proc.yang**: /proc file system monitoring information. Will be available if WITH\_YUMA\_PROC=1 build variable is set.
- **yuma-arp**: collection of YANG definitions for configuring and monitoring ARP. Will be available if WITH\_YUMA\_ARP=1 build variable is set.
- **ietf-restconf-monitoring**: monitoring information for the RESTCONF protocol. Will be available if WITH\_RESTCONF=1 build variable is set.

## 2.2 Intended Audience

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

It contains the following information:

- Introduction to YANG and NETCONF based Network Management
- YumaPro Configuration
- YumaPro XPath Reference
- YumaPro Error Reference

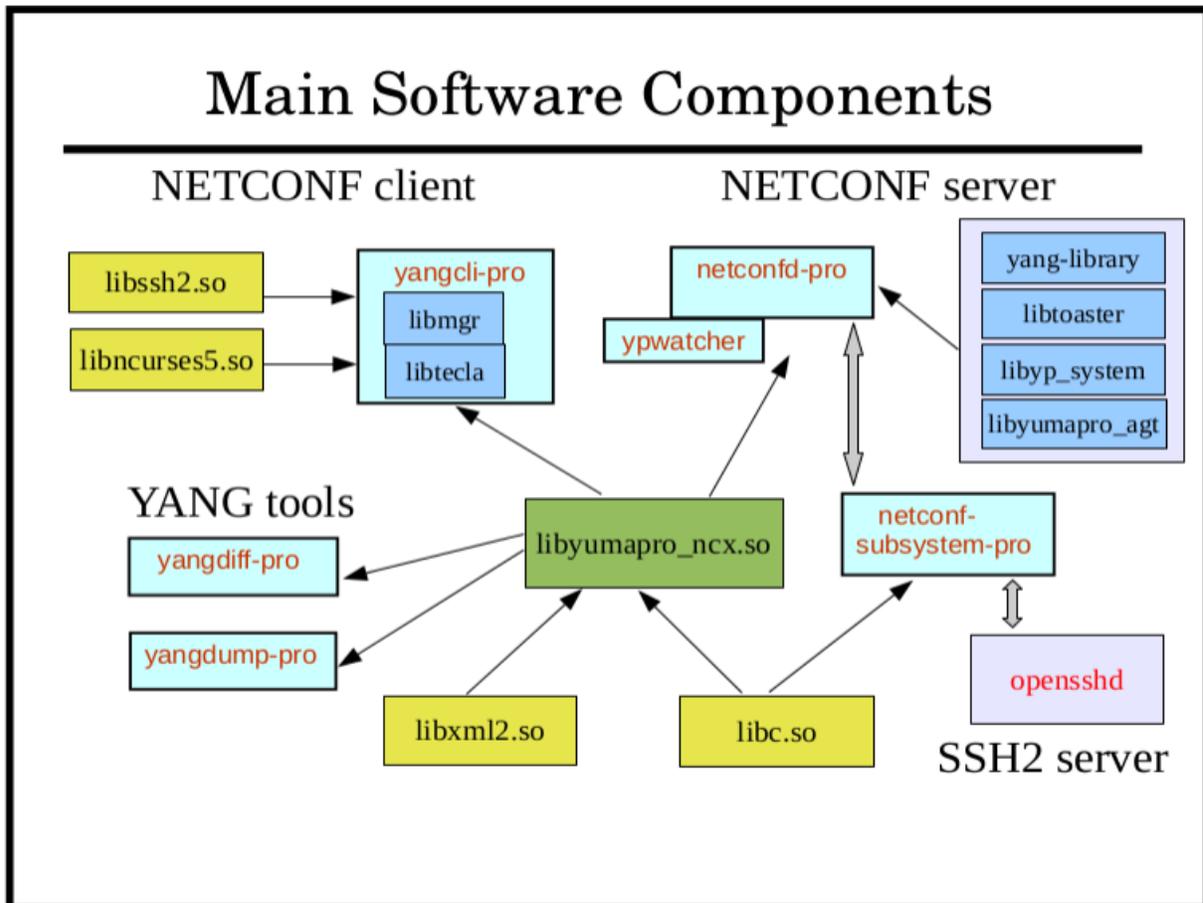
## 3 Introduction

The YumaPro 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 YumaPro, and needs to be pre-installed:

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

The following external libraries are used by YumaPro, 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 (libncurses5)**

## YumaPro User Manual

- Curses terminal support (needed on Fedora platforms only)
- **libxml2**
  - xmlTextReader XML parser
  - pattern support

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

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

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

- **libyumapro\_ncx**
  - YANG parser
  - YANG validation
  - basic NETCONF support
  - XPath support
  - configuration database support

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

- **libyumapro\_agt**
  - NETCONF server support, shared with SIL code
- **libmgr**
  - NETCONF client support
- **libycli (not shown)**
  - NETCONF client support used by yangcli-pro and yp-shell
- **libydump (not shown)**
  - yangdump-pro translation functionality

The following binaries are built by YumaPro:

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

- **yang-api (not shown)**
  - FastCGI application used to integrate the netconfd-pro server with a WEB server for YANG-API
- **yp-shell (not shown)**
  - client shell program used to provide CLI access to netconfd-pro server
- **ypwatcher**
  - Automatic server state monitoring program.
- **restconf (not shown)**
  - FastCGI application used to integrate the netconfd-pro server with a WEB server for RESTCONF

The following sample netconfd-pro module instrumentation library is provided as an example. These libraries can only be created with the YumaPro SDK. Refer to the YumaPro Developer's Guide for details on creating server instrumentation libraries.

- **libtoaster**
  - Server instrumentation code for the YANG module libtoaster.yang.
- **libyp\_system**
  - Example system integration library used with netconfd-pro

### 3.1.1 YANG

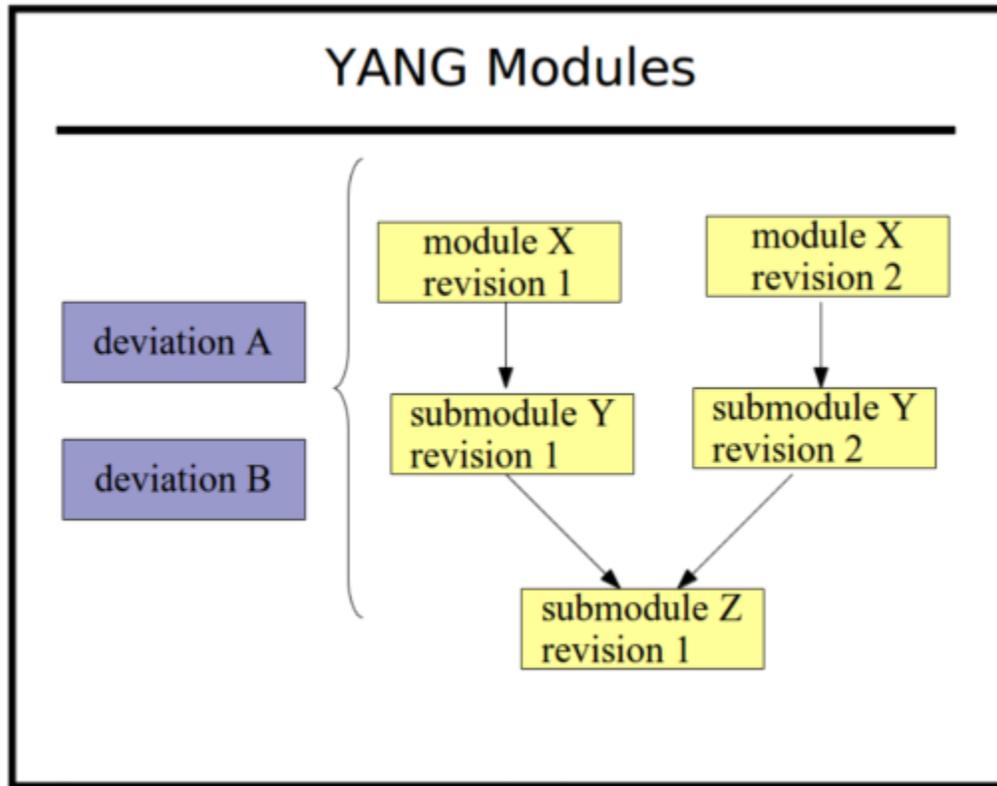
A YANG module define the semantics and syntax of a specific management feature. They are similar to SMIV2 (MIB) modules, but much more powerful and extensible. YANG provides the ability to define a detailed programmatic interface utilizing all protocol features:

- 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 YumaPro 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.

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'.

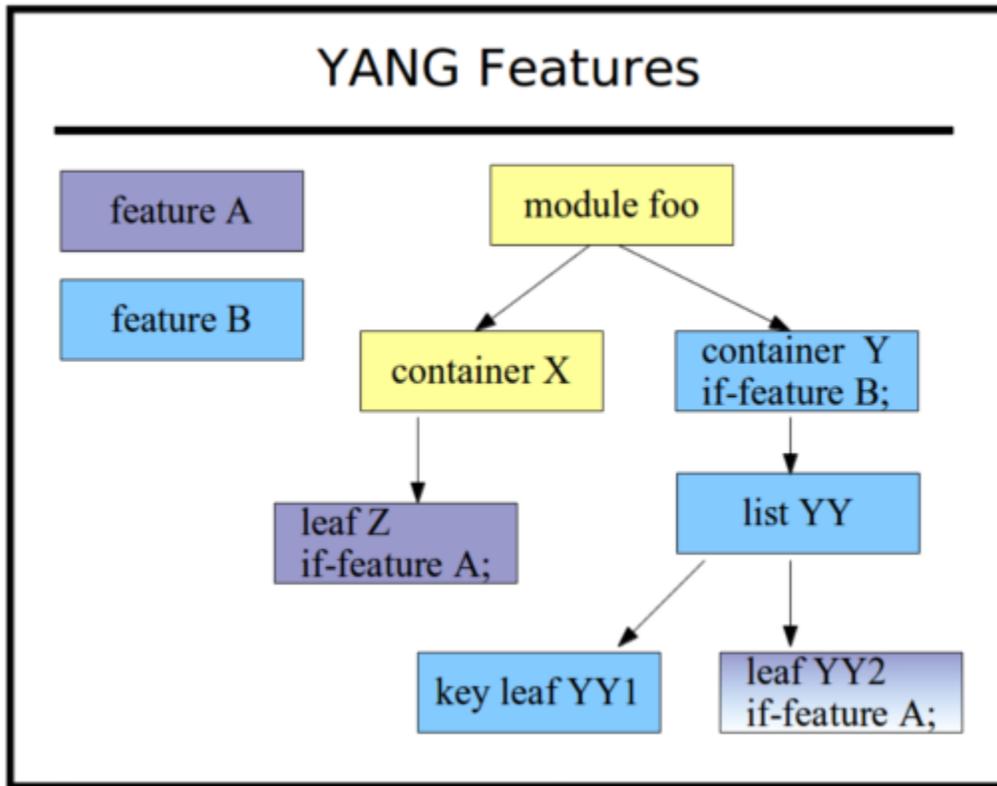
YumaPro programs support concurrent usage of different revisions of the same module or submodule. This can occur via groupings from external modules within the YANG language. Only one revision of a module can be imported into a single module or submodule, but any of these files may in turn import other 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, YumaPro 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.

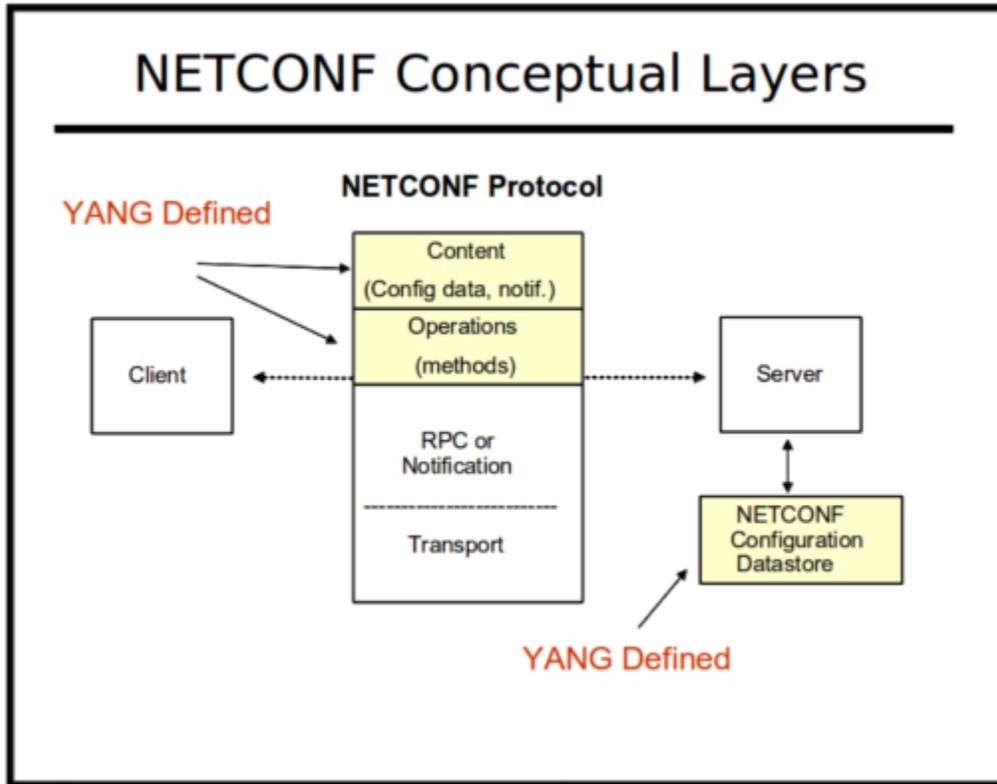
Objects that contain the if-feature statement for the corresponding feature are part of the feature.

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.

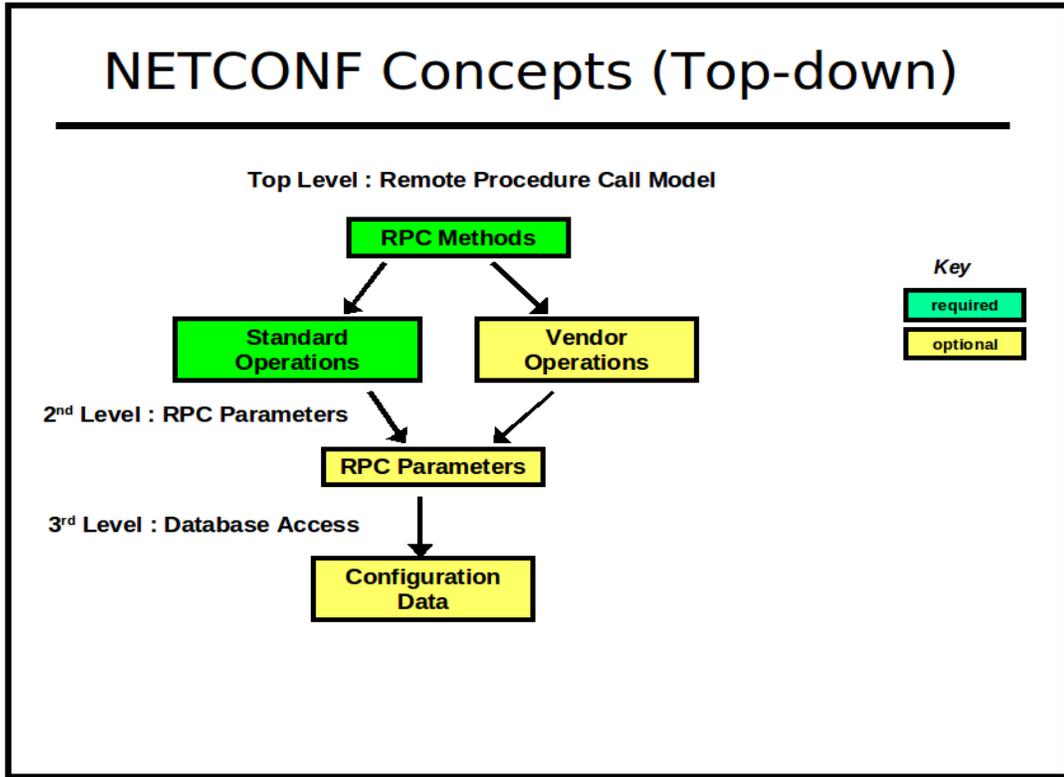
Multiple if-feature statements form a logical AND expression. All the referenced features must be enabled for the object to be available. In the example above, leaf 'YY2' is not present unless feature A and B are both advertised by the server.

### 3.1.2 NETCONF

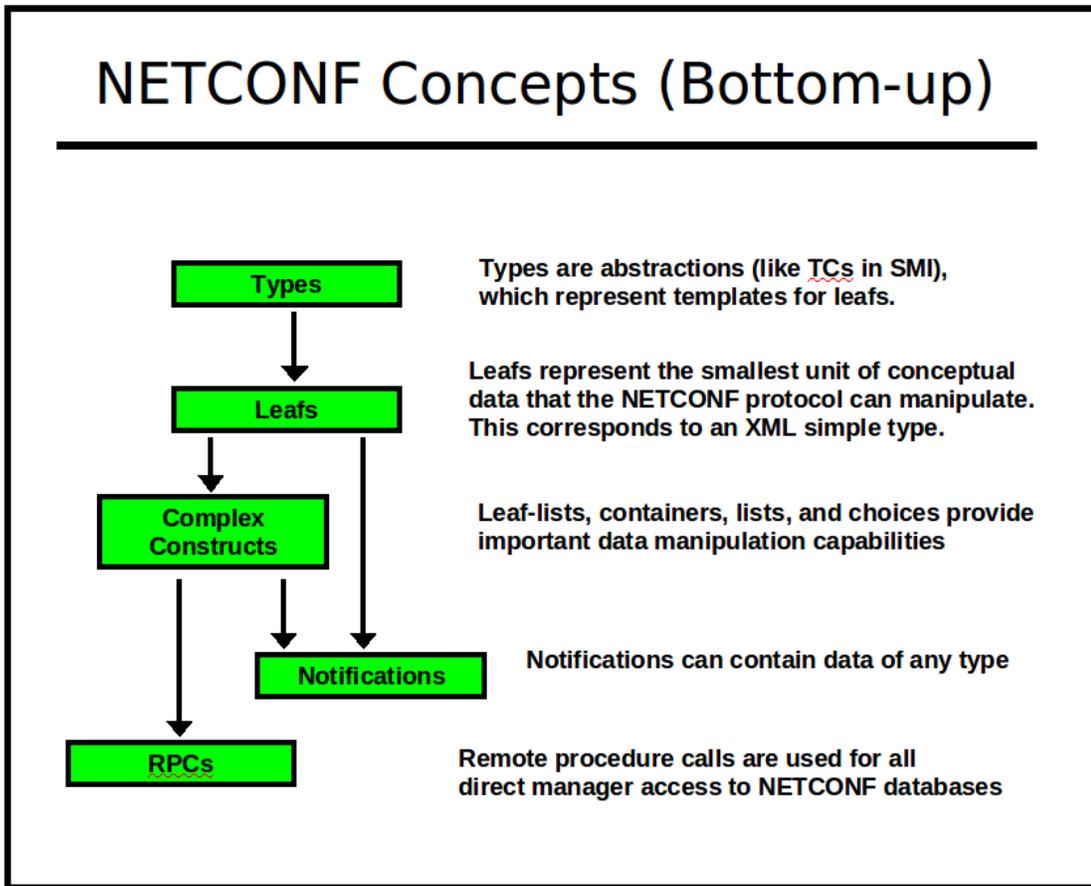
The mandatory components of the NETCONF protocol are defined in RFC 6241 and RFC 6242.



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 YumaPro. The **OpenSSH** server is used in the **netconfd-pro** implementation, and **libssh2** library is used in the **yangcli-pro** implementation, to provide all SSH2 layer support.

By default, TCP port 830 (netconf-over-ssh) is used for all NETCONF communications between **yangcli-pro** and **netconfd-pro**. 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.

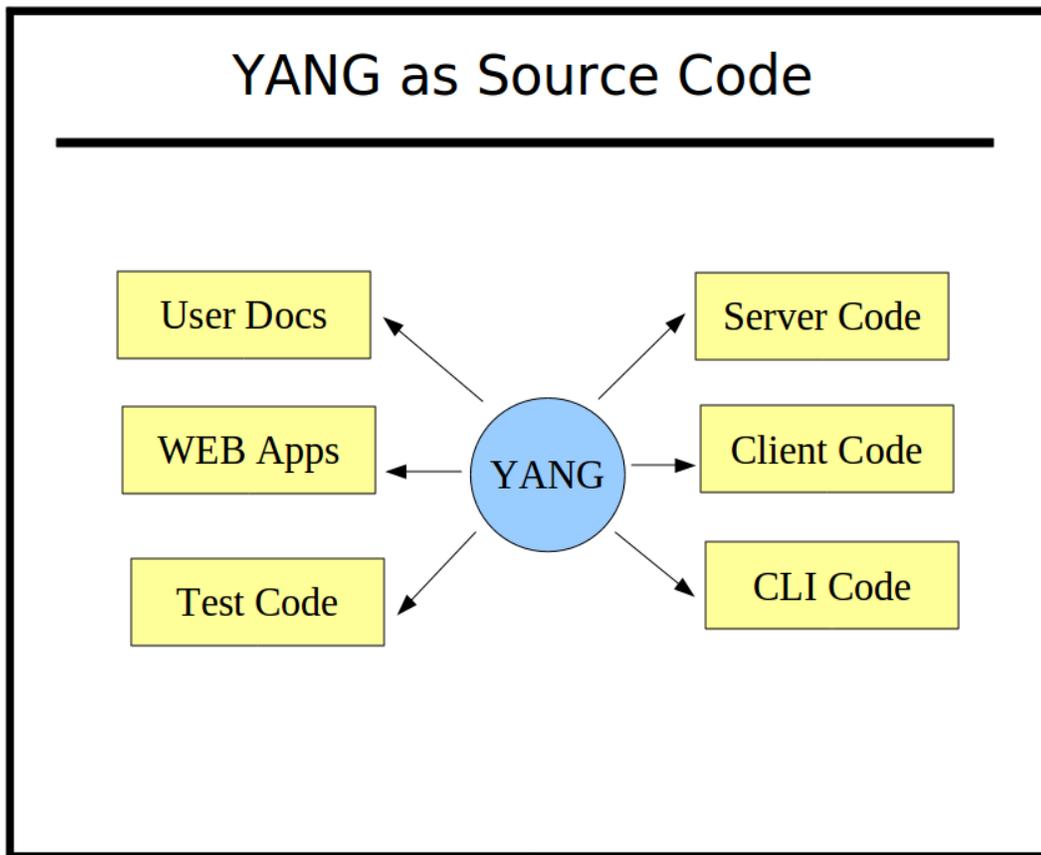
Access control is configurable (via **ietf-nacm.yang** or **yuma-nacm.yang**), based on group membership. The access control rules permit or deny access to one or more groups, to a subset of the YANG content. Separate defaults for read, write, and exec (RPC operation) access are provided.

### 3.1.3 YANG-based Automation

YumaPro 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 implementation.

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-pro** client and **netconfd-pro** 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**

<b>YANG statement</b>	<b>description</b>
<b>config</b> <i>boolean</i> ;	The <b>config</b> 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.
<b>default</b> <i>string</i> ;	The <b>default</b> 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 <b>typedef</b> or <b>leaf</b> statements. If both are defined, then the leaf default will be used.
<b>YANG statement</b>	<b>description</b>
<b>deviation</b> <i>deviation-target-path</i> { ... }	The <b>deviation</b> 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 <b>deviation</b> 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.
<b>error-app-tag</b> <i>apptag-string</i> ;	The <b>error-app-tag</b> statement can be used within the <b>range</b> , <b>length</b> , and <b>pattern</b> 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.
<b>error-message</b> <i>errmsg-string</i> ;	The <b>error-message</b> statement can be used within the <b>range</b> , <b>length</b> , and <b>pattern</b> 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.
<b>extension</b>	The <b>extension</b> 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.
<b>feature</b>	The <b>feature</b> 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.
<b>if-feature</b> <i>feature-name</i> ;	Construct containing the <b>if-feature</b> statement is only included if the specified feature is supported by the server. Otherwise, the object does not exist on the server.
<b>import</b> (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.

## YumaPro User Manual

<b>include</b> (by revision)	The <b>include</b> statement provides the exact same features as the <b>import</b> 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.
<b>key</b> <i>key-leaf-list</i> ;	The <b>key</b> 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.
<b>length</b> <i>length-spec-string</i> ;	The <b>length</b> statement is exactly like the <b>range</b> statement, except it limits the length of string <b>leaf</b> and <b>leaf-list</b> objects.
<b>YANG statement</b>	<b>description</b>
<b>mandatory</b> <i>boolean</i> ;	The <b>mandatory</b> 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.
<b>max-elements</b> <i>number</i>   'unbounded' ;	Specifies the maximum number of instances that a <b>list</b> or <b>leaf-list</b> object can have in a valid database. The default is 'unbounded', if this statement is not present.
<b>min-elements</b> <i>number</i> ;	Specifies the minimum number of instances that a <b>list</b> or <b>leaf-list</b> object must have in a valid database. The default is zero, if this statement is not present.
<b>must</b> <i>xpath-expr</i> ;	If the object containing the <b>must</b> statement exists, then the XPath expression must evaluate to 'true' for the database to be valid. This provides referential integrity checks among related parameters.
<b>pattern</b> <i>pattern-string</i> ;	The <b>pattern</b> statement specifies a regular expression that must evaluate to 'true' in order for the corresponding string <b>leaf</b> or <b>leaf-list</b> object to be valid. Multiple patterns encountered in a nested typedef chain must all evaluate to 'true' for the object to be valid.
<b>range</b> <i>range-spec-string</i> ;	The <b>type</b> 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.
<b>refine</b> <i>refine-target-path</i> { ... }	The <b>refine</b> statement is defined within a <b>uses</b> 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 <b>refine</b> statement.
<b>revision</b> <i>revision-date</i> { ... }	The <b>revision</b> statement identifies the most current version of a YANG module or sub-module. Multiple versions at once are supported in YANG.

## YumaPro User Manual

<b>unique</b> <i>unique-node-list</i> ;	The <b>unique</b> 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.
<b>uses</b> <i>grouping-name</i> ;	The <b>uses</b> statement inserts an instance of a reusable <b>grouping</b> , replacing the uses node within the conceptual data tree.
<b>when</b> <i>xpath-expr</i> ;	The object containing the <b>when</b> 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 YumaPro. 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 YumaPro 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 occur.

Most of these extensions apply to **netconfd-pro** server behavior, but not all of them. For example, the **ncx:hidden** extension will prevent **yangcli-pro** from displaying help for an object containing this extension. Also, **yangdump-pro** 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 YumaPro, if encountered in a YANG file:

**YANG Language Extensions**

<b>extension</b>	<b>description</b>
<b>ncx:hidden;</b>	Declares that the object definition should be hidden from all automatic documentation generation. Help will not be available for the object in <b>yangcli-pro</b> .
<b>ncx:metadata</b> “ <i>attr-type</i> <i>attr-name</i> ”;	Defines a qualified XML attribute in the module namespace. Allowed within an RPC input parameter. <b>attr-type</b> is a valid type name with optional YANG prefix. <b>attr-name</b> is the name of the XML attribute.
<b>ncx:no-duplicates;</b>	Declares that the <b>ncx:xsdlist</b> data type is not allowed to contain duplicate values. The default is to allow duplicate token strings within an <b>ncx:xsdlist</b> value.
<b>ncx:password;</b>	Declares that a string data type is really a password, and will not be displayed or matched by any filter.
<b>ncx:qname;</b>	Declares that a string data type is really an XML qualified name. XML prefixes will be properly generated by <b>yangcli-pro</b> and <b>netconfd-pro</b> .
<b>ncx:root;</b>	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.
<b>ncx:schema-instance;</b>	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 <b>nacm</b> <data-rule> expressions.
<b>nacm:secure;</b>	Declares that the database object is a secure object. If the object is an <b>rpc</b> statement, then only the <b>netconfd-pro</b> '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.
<b>extension</b>	<b>description</b>
<b>ncx:user-write &lt;bits&gt;;</b>	Declares the user write permissions that will be allowed for a config=true data node. The bits parameter contains the write permissions that will be allowed for the object (create, update, delete).
<b>nacm:very-secure;</b>	Declares that the database object is a very secure object. Only the 'superuser' will be allowed to access the object, by default.
<b>ncx:xsdlist "list-type";</b>	Declares that a string data type is really an XSD style list. <b>list-type</b> is a valid type name with optional YANG prefix. List processing within <edit-config> will be automatically handled by <b>netconfd-pro</b> .
<b>ncx:xpath;</b>	Declares that a string data type is really an XPath expression. XML prefixes and all XPath processing will be done automatically by <b>yangcli-pro</b> and <b>netconfd-pro</b> .

### 3.1.5 YANG Compiler

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

The complete YANG language is supported, as defined in RFC 6020. 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 YumaPro programs, including netconfd-pro server instrumentation. Refer to the 'YANG User Guide' for details on writing YANG files and using the extensions built into YumaPro.

Note: The **smidump** is not part of YumaPro, but it can be utilized to convert MIB modules written in SMiv2 into YANG modules, which can then be implemented in **netconfd-pro**, and managed with **yangcli-pro**. 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.

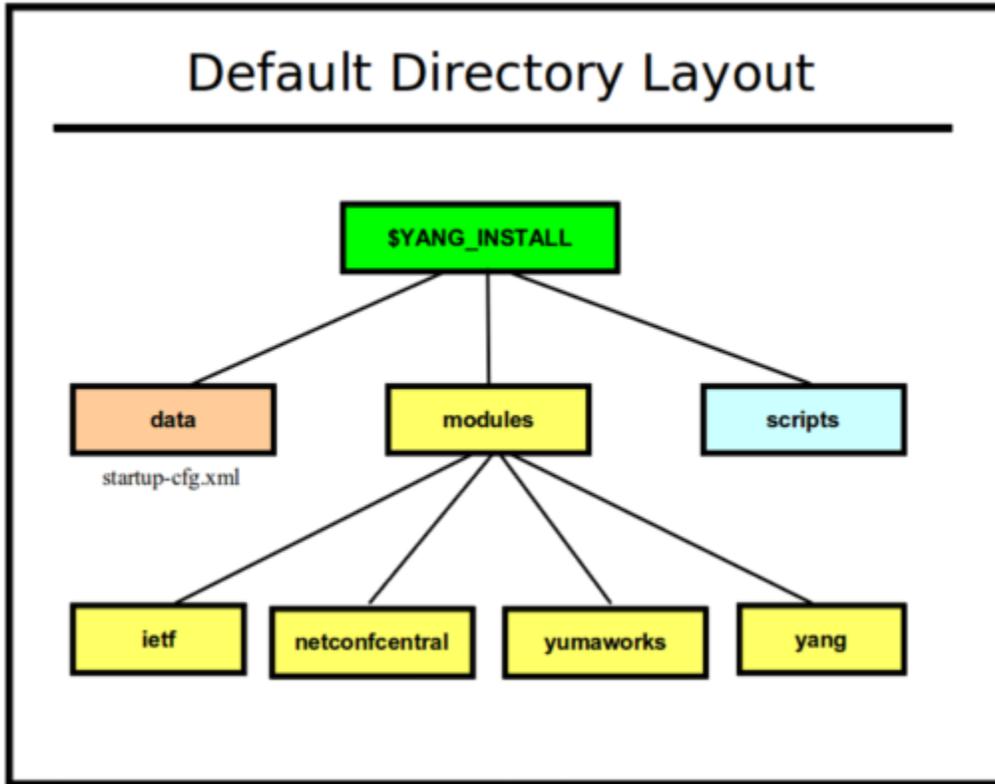
### Directory Layout

YumaPro 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 (YUMAPRO\_INSTALL) shared by all users, and one or more 'project' roots (YUMAPRO\_HOME), which can be shared but may belong to a single user.

The YumaPro 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-pro**
- command-line-history file for **yangcli-pro**

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

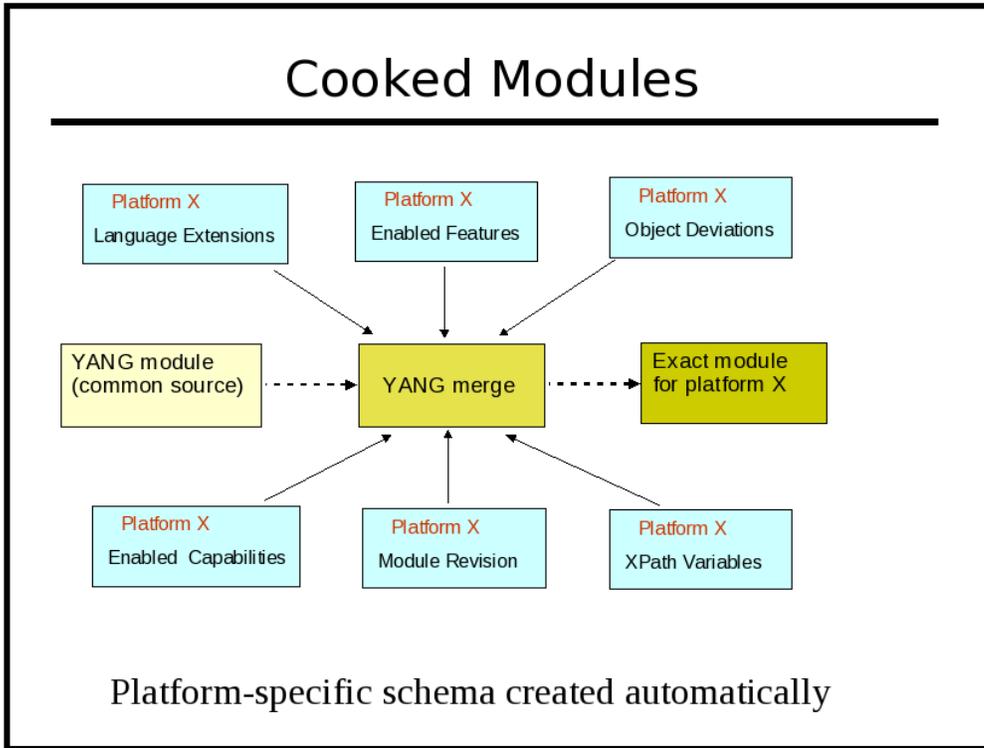


**Module Revisions**

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-pro** 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.

YumaPro 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 YumaPro 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 YumaPro 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-pro** program. Many 'reports' can also be generated:

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

The **yangdump-pro** 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 <div>** 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-pro** 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.

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

### 3.1.9 NETCONF Servers

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.

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

## 4 System Configuration

The YumaPro 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-pro {
  ncx:cli;
  // yangcli-pro CLI parameters defined as choices and leafs
  here
}
```

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

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

Note:

- The **netconf-subsystem-pro** 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-pro** program for command line editing support, has its own configuration file `~/.tecla`, to override the default (emacs) editing key assignments.

YumaPro 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 YumaPro 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 YumaPro:

- HOME
- YUMAPRO\_HOME
- YUMAPRO\_INSTALL
- YUMAPRO\_MODPATH
- YUMAPRO\_LOADPATH
- YUMAPRO\_DATAPATH
- YUMAPRO\_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 YumaPro 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 \$YUMAPRO\_HOME

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

Default value: none

CLI override: --yuma-home

CLI example:

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

C shell example:

```
setenv YUMAPRO_HOME /home/andy/swdev/yumapro/trunk/netconf
```

Bash shell example:

```
export YUMAPRO_HOME=/home/andy/swdev/yumapro/trunk/netconf
```

### 4.1.3 \$YUMAPRO\_INSTALL

The `$YUMAPRO_INSTALL` environment variable contains the directory specification of the YumaPro installation root directory.

Default value: `/usr/share/yumapro`

CLI override: none

C shell example:

```
setenv YUMAPRO_INSTALL /sw/yumapro
```

Bash shell example:

```
export YUMAPRO_INSTALL=/sw/yumapro
```

#### 4.1.4 \$YUMAPRO\_MODPATH

The **\$YUMAPRO\_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**, **\$YUMAPRO\_HOME**, and **\$YUMAPRO\_INSTALL** environment variables, which specify a YumaPro root directory.

Default value: none

CLI override: **--modpath**

CLI example:

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

C shell example:

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

Bash shell example:

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

## 4.1.5 \$YUMAPRO\_LOADPATH

The **\$YUMAPRO\_LOADPATH** environment variable contains a list of directory specifications that should be searched (in order) to load YANG or YIN modules into the server. It is supported by netconfd-pro only. It can be used to simplify loading of YANG modules.

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**, **\$YUMAPRO\_HOME**, and **\$YUMAPRO\_INSTALL** environment variables, which specify a YumaPro root directory.

Default value: none

CLI override: `--loadpath`

CLI example:

```
--loadpath="$HOME/modules2"
```

C shell example:

```
setenv YUMAPRO_LOADPATH "$HOME/modules2"
```

Bash shell example:

```
export YUMAPRO_LOADPATH="$HOME/modules2"
```

## 4.1.6 \$YUMAPRO\_DATAPATH

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

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

The location where the **netconfd-pro** 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**, **\$YUMAPRO\_HOME**, and **\$YUMAPRO\_INSTALL** environment variables, which specify a YumaPro root directory.

Default value: none

CLI override: **--datapath**

CLI example:

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

C shell example:

```
setenv YUMAPRO_DATAPATH "$HOME/mydata:$HOME/project1/data"
```

Bash shell example:

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

### 4.1.7 \$YUMAPRO\_RUNPATH

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

Script files used by the **yangcli-pro** 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**, **\$YUMAPRO\_HOME**, and **\$YUMAPRO\_INSTALL** environment variables, which specify a YumaPro root directory.

Default value: none

CLI override: **--runpath**

CLI example:

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

C shell example:

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

Bash shell example:

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

## 4.2 Searching for Files

All YumaPro 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 YumaPro 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
```

## YumaPro User Manual

- 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  
$YUMAPRO_HOME/data/startup-cfg.xml
```

Note: Whenever YumaPro 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
- \$YUMAPRO\_HOME
- \$YUMAPRO\_MODPATH
- \$YUMAPRO\_LOADPATH
- \$YUMAPRO\_DATAPATH
- \$YUMAPRO\_RUNPATH

The following configuration parameters affect file searches:

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

## 4.2.1 YumaPro Work Directory

There is a directory (**\$HOME/.yumapro**) created by **yangcli-pro** or **netconfd-pro** for data files and temporary files. It is called **.yumapro**, 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-pro**, 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-pro\_history** file for **yangcli-pro** command line history recall.

The **\$HOME/.yumapro/tmp** directory is used by **yangcli-pro** 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-pro**.

## 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-pro** 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

YumaPro 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 exhausted 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 **\$YUMAPRO\_LOADPATH** environment variable, or set with the **--loadpath** 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. Each directory specified in the **\$YUMAPRO\_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.
4. 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.
5. The **\$YUMAPRO\_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.
6. The **\$YUMAPRO\_INSTALL/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.

### Data Search Path

YumaPro programs can store data used during operation.

## YumaPro User Manual

An example of a data file is the startup configuration file used by **netconfd-pro**, 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 **\$YUMAPRO\_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 **\$YUMAPRO\_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/.yumapro** directory is checked.
7. The **\$YUMAPRO\_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/yumapro/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/yumapro** directory is checked.

### **Script Search Path**

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

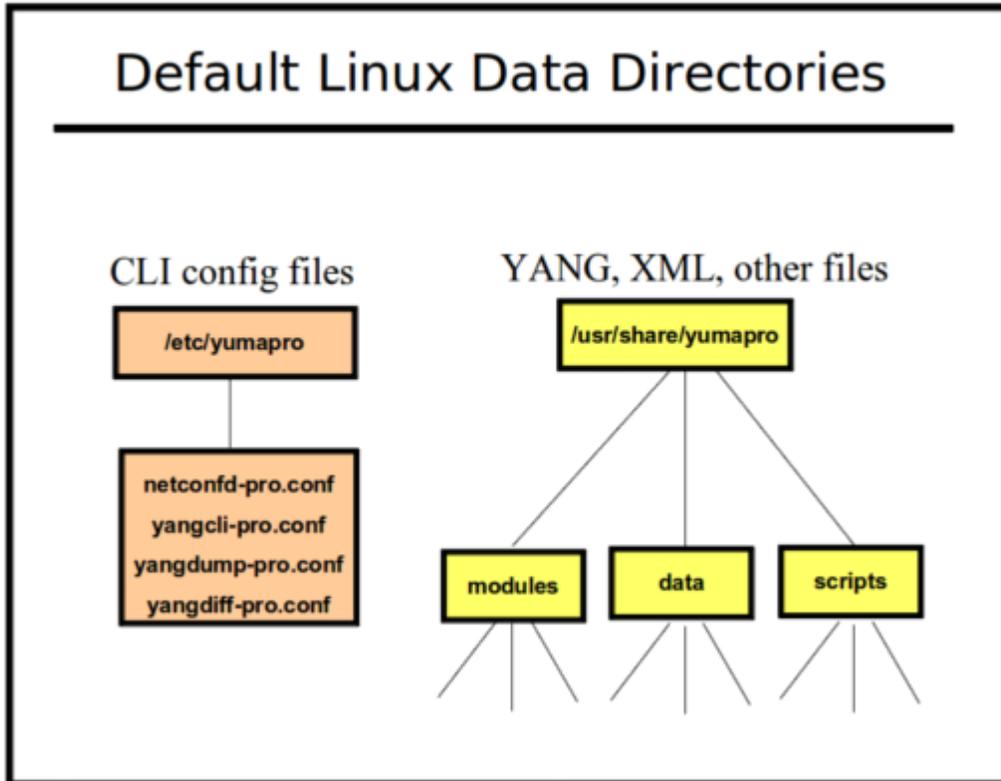
1. If the file name has an absolute path specification, then that exact file location is tried. If no match is found, then the search will terminate in failure.
2. The current working directory is checked. No sub-directories are checked, if any are present.
3. Each directory specified in the **\$YUMAPRO\_RUNPATH** 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 **\$YUMAPRO\_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 **\$YUMAPRO\_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 YumaPro 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-pro** 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-pro**:

- --indent
- --with-defaults

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

- --datapath
- \$YUMAPRO\_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-pro">
  <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 YumaPro 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-pro {
  # 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
}
```

## YumaPro User Manual

- 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-pro {
    # common yangdump-pro parameters here
}

yangdiff-pro {
    # common yangdiff-pro 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 Logging

All YumaPro programs are capable of generating informational messages of various kinds. By default, these will be directed to STDOUT. With the addition of specific startup parameters, the output may instead be directed to either a file, or to a “syslog” or vendor-specific daemon (if configured on the local system). Depending on the exact set of parameters specified, varying output may be directed to a combination of these “streams”.

Message types include error, warning, general information, and debugging related detail.

The following “levels” are defined (in order from “lowest” to “highest”):

- **error**
- **warn**
- **info**
- **debug**
- **debug2** (more detailed)
- **debug3** (even more detailed)
- **debug4** (most detailed)

These keywords may be specified to the **log-level** command, either via the command line (see bootstrap CLI below) or the config file:

- **--log-level=debug** (command line)
- **log-level debug** (config file)

By default, without any explicit configuration, all programs use “**info**” as the log level. The log level values are “ordered” and “cumulative”, meaning that log messages will be generated at all levels up to and including the specified level. For example, if **--log-level=debug** is specified on the command line, error, warning, info and debug level messages will be output. Debug2, debug3, and debug4 level messages will be suppressed.

Note that netconfd-pro supports all log commands described below. Yangcli-pro support is similar with the exception of thread-specific commands. Yangdiff-pro and Yangdump-pro support only the log, log-append, and log-level commands.

Logging related commands may be grouped roughly into 3 categories:

- Output Stream – where to direct log message output.
- Detail – what supporting information to include.
- Debugging and Development – information useful to developer debugging and bug reporting.

### 4.4.1 Logging Output Stream Commands

Note: In the examples below, command line CLI syntax will be used. Equivalent functionality can be achieved through using a config file instead.

By default, log message output is directed to STDOUT. If the **--log** command is used to specify a valid filename, messages are sent to a file instead. If **--log-syslog** is specified, messages are sent to the local syslog daemon. If **--log** and **--log-syslog** are both specified, both streams receive log message output. To display the output again via STDOUT, add the **--log-console** command.

There is an additional customer-specific logging capability analogous to **--log-syslog**. Syslog output may be directed to a customer-written and registered callback function by using the **--log-vendor** capability.

- **--log=<filename>**: Send log message output to <filename> instead of STDOUT.
- **--log-append**: Append to existing log file without first truncating it.
- **--log-syslog**: Direct all logger output to the syslog daemon.
- **--log-console**: If **--log** and/or **--log-syslog** are present, duplicate logger output via STDOUT.
- **--log-mirroring**: Synonym for **--log-console**.
- **--log-stderr**: If STDOUT is in use, direct “**error**” level log messages to STDERR instead.
- **--log-vendor**: If present, log messages will be directed to a customer-written and registered callback function. (In the absence of a registered callback, this parameter will direct logging messages to syslog. This facilitates stand-alone testing.) Relative to the different output streams, this parameter behaves like **--log-syslog** as described above. Note that **--log-syslog** and **--log-vendor** are mutually exclusive. Enabling this functionality is explained in more detail by an API specified in the YumaPro API Reference Manual.

### 4.4.2 Logging Detail Commands

- **--log-level=<level>**: Output standard log messages up to and including the specified level. Valid level keywords are those listed above in Section 4.4. By default, the **info** level is used.
- **--log-pthread-level=<level>**: Output thread-specific log messages up to and including the specified level. (Note that these messages are generated in images with pthreads support only). Valid level keywords are those listed above in Section 4.4. By default, the info level is used. This output will appear in the syslog output stream only if allowed by the applicable stream filter (i.e., **--log-syslog-level**).
- **--log-syslog-level=<level>**: Output syslog-directed messages up to and including the specified level. Valid level keywords are those listed above in Section 4.4. By default, the info level is used.

Syslog stream output is now filtered via this directive. This means, for example, that syslog output could be limited to standard operational “info” level messages while a log file was used to capture “debug” level output.

- **--log-header=”<keyword-list>”**: Prepend log-level and date/time information to each log message. Valid keywords include:
  - **custom** – Date/time information is in YANG canonical format.
  - **localtime** – Date/time information is in local time.

Multiple keywords may be specified. Keyword(s) should always be enclosed by double quotes and separated by spaces.

Note that syslog output does not receive this treatment since syslog adds its own header information, based on how it is configured.

### 4.4.3 Logging Debug and Development Commands

Generally these commands are used internally, as well as by developers with a source code license. In addition using one or more of the commands in this group may be requested by customer support in order to help with debugging issues in the field. Note: backtrace-related functionality is not supported by Windows or Mac versions of the software at this time.

- **--log-backtrace=<max-frames>**: Append stack trace information to each log message. Display a maximum of <max-frames> call frames. Use the value 0 to specify the internal default (20).
- **--log-backtrace-level="<keyword-list>"**: If **log-backtrace** is present, generate stack trace detail for the specified log level(s) only. Valid keywords include those listed above for **log-level**. Multiple level keywords may be specified. Keyword(s) should be enclosed by double quotes and separated by spaces.
- **--log-backtrace-detail**: Generate additional, more detailed stack trace information (specifics depend upon compiler and host operating system).
- **--log-backtrace-stream="<keyword-list>"**: If **log-backtrace** is present, limit stack trace detail for the specified log message streams only. Valid keywords include:
  - **logfile**
  - **stdout**
  - **stderr**
  - **syslog**
  - **vendor**

Multiple keywords may be specified. Keyword(s) should always be enclosed by double quotes and separated by spaces.

Note that in order to provide stack trace information to the GNU GDB debugger, a special image build may be required (with debug symbols and statically linked libraries). Specific requirements are described in the **Using Backtrace Information** section below.

#### 4.4.4 Logging Examples

Note: The following examples use the command line version of the syntax (see the Bootstrap CLI section below for more information).

By default, logging output goes to STDOUT:

```
mydir> netconfd-pro --superuser=marco --log-level=debug
...
Running netconfd-pro server (5.0.aa088bf)
...
New session 1 created OK
Session 1 for marco@127.0.0.1 now active (base:1.1)
...
```

Log output may be redirected to a file instead of STDOUT:

```
mydir> netconfd-pro --superuser=marco --log-level=debug \
>> output.log
```

Error level log output may be directed to STDERR even while other log output is redirected to a file:

```
mydir> netconfd-pro --superuser=marco --log-level=info \
--log-stderr --logX >> output.log

Error: Unknown parameter (logX)
```

Logging output may also be directed to a file using the `--log=<filename>` CLI command. In this case, all output is sent to the log file (including error level messages). No output will appear on STDOUT beyond the location and name of the newly opened log file. The `--log-stderr` directive, if specified, is a NOP:

```
mydir> netconfd-pro --superuser=marco --log-level=info \
--log=logfile
```

Logging output may be directed instead to SYSLOG. In this case, netconfd-pro messages are designated as "daemon" level, while yangcli-pro messages are designated as "user" level output. The YumaPro debug level of each message is translated into an appropriate SYSLOG priority level:

## YumaPro User Manual

```
mydir> netconfd-pro --superuser=marco --log-level=info \  
--log-syslog
```

No output will be directed to STDOUT, STDERR, or a log file.

The presence of `--log` and `--log-syslog` commands together will cause log output to be sent to the specified log file after the message is sent for SYSLOG processing.

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
--log-syslog -log=logfile
```

No output will be directed to STDOUT or STDERR

The addition of the `-log-console` directive will cause log output to be displayed via STDOUT, after the message is sent for SYSLOG and/or log file processing:

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
--log-syslog -log=logfile --log-console
```

The presence of `--log` and `--log-syslog` commands together will cause log output to be sent to the specified log file after the message is sent for SYSLOG processing. It is possible to include different debug levels in the two different output streams by using the `--log-syslog-level` filter. The following will send debug level messages to the log file, but restrict syslog content to info level message output:

```
mydir> netconfd-pro --superuser=marco -log=logfile \  
-log-level=debug --log-syslog -log-syslog-level=info
```

No output will be directed to STDOUT or STDERR

Logging output to STDOUT, STDERR, or a log file, whether direct or "mirrored" may be "customized" with date, time, and debug level information. This is the purpose of the `--log-header="custom"` directive. The date/time string may be changed from the Yang default "Zulu" (UCT) format to use local time by including the `localtime` keyword.

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
-log-header="custom"
```

or

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
--log-header="custom localtime"
```

It is also possible to include call stack trace information (“backtrace”) within the logging message stream (including SYSLOG). For example:

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
--log-syslog --log=file.log --log-backtrace=0
```

Will display as many as 20 stack frames of backtrace information (default). Examples of output generated by the command above follow:

Log file:

```
agt_ses msg ready for session 1  
--Backtrace: [0x809ee3b][0x80760d1][0x806adbc][0x805d0a4][0x805d26d][0xb74af113][0x805cbf1]
```

SYSLOG:

```
[debug] [daemon] [Jul  5 19:03:37] netconfd-pro[7576]:### agt_ses msg ready for session 1  
###  
[info] [daemon] [Jul  5 19:03:37] netconfd-pro[7576]:### --Backtrace: [0x80760d1][0x806adbc]  
[0x805d0a4][0x805d26d][0xb74af113][0x805cbf1] ###
```

Given the above example, with logging info being directed via multiple streams, it is possible to restrict backtrace output to say, just the SYSLOG file, or just the console or log file:

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
--log-syslog --log-console --log-backtrace=33 \  
--log-backtrace-stream="stdout"
```

In the above example, backtrace info appears only on the console stream. By contrast, backtrace info will only be included in the SYSLOG file in the next example:

```
mydir> netconfd-pro --superuser=marco --log-level=debug \  
--log-syslog --log-console --log-backtrace=33 \  
--log-backtrace-stream="syslog"
```

Production of backtrace info can be restricted to one or more debugging levels with the addition of the **log-backtrace-level** directive:

## YumaPro User Manual

```
mydir> netconfd-pro --superuser=marco --log-level=debug2 \  
        --log-syslog --log-console --log-backtrace=33 \  
        --log-backtrace-level="error warn debug2"
```

Any of error, warning, or debug2 level log entries displayed on either the console or via syslog will include backtrace detail. Info and other debug level messages would NOT include backtrace detail.

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

## 4.4.5 Logging Backtrace Information

Under special circumstances, YumaWorks customer support may request that an image be run with the **--log-backtrace** enabled for trouble-shooting purposes. In such a case, the resulting log file would be forwarded to YumaWorks for analysis.

For source code development customers only, it is possible to cut and paste information from the output of the **--log-backtrace** command directly into the **gdb** debugger in order to construct a symbolic stack trace corresponding to the output of a given log message. Note that at the present time, a statically linked image containing debug symbols must be used. Consider the following example:

- The image being run is a “debug static” image (built via “**make DEBUG=1 STATIC=1 ...**”).
- The image might be run as follows:

```
mydir> netconfd-pro --superuser=user1 --log=log-001.log \
        --log-header="custom localtime" \
        --log-backtrace=0 --log-level=debug
```

- You run **yangcli-pro**. After connecting to the server, a sequence of commands is issued. In the log file, you notice the following warning message:

```
[2012-10-11 17:46:59] [warn] Warning: SIL code for module 'test-xxx' not found
--Backtrace: [0x427f7b][0x430fa6][0x43a02b][0x434c8a][0x42a06d][0x41aca7][0x41ae28]
[0x7f8a0656476d][0x41a799]
```

- In order to decode the stack backtrace, in another window, **cd** to the build directory containing the sources that correspond to the image. Run **gdb** like so:

```
mydir> gdb /usr/sbin/netconfd-pro
```

- Now cut and paste individual hex addresses from the backtrace output into the **gdb** window, after typing the “**list \***” command at the **gdb** prompt. The result should look something like this for the first three frames:

```
mydir> gdb /usr/sbin/netconfd-pro
```

```
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /usr/sbin/netconfd-pro...done.
```

```

(gdb) list *0x427f7b
0x427f7b is in load_invoke (agt_ncx.c:2938).
2933             (revval) ? VAL_STR(revval) : NULL,
2934             TRUE);
2935         if (res == ERR_NCX_SKIPPED) {
2936             log_warn("\nWarning: SIL code for module '%s' not found",
2937                 VAL_STR(modval));
2938             res = ncxmod_load_module(VAL_STR(modval),
2939                 (revval) ? VAL_STR(revval) : NULL,
2940                 &agt_profile->agt_savedevQ,
2941                 &mod);
2942         } else if (res == NO_ERR) {
(gdb) list *0x430fa6
0x430fa6 is in agt_rpc_dispatch (agt_rpc.c:1617).
1612     * especially since the return of data can be automated
1613     * in the send_rpc_reply phase.
1614     */
1615     if ((res==NO_ERR) && cbset && cbset->acb[AGT_RPC_PH_INVOKE]) {
1616         msg->rpc_agt_state = AGT_RPC_PH_INVOKE;
1617         res = (*cbset->acb[AGT_RPC_PH_INVOKE])(scb, msg, &method);
1618         if (res != NO_ERR) {
1619             /* make sure there is an error recorded in case
1620              * the invoke phase callback did not add one
1621             */
(gdb) list *0x43a02b
0x43a02b is in agt_top_dispatch_msg (agt_top.c:153).
148
149     /* check node type and if handler exists, then call it */
150     if (top.nodetyp==XML_NT_START || top.nodetyp==XML_NT_EMPTY) {
151         /* find the owner, elname tuple in the topQ */
152         handler = top_find_handler(top.module, top.elname);
153         if (handler) {
154             /* call the handler */
155             (*handler)(scb, &top);
156         } else {
157             res = ERR_NCX_DEF_NOT_FOUND

```

## 4.4.6 Logging Hints

This section will be expanded over time as tips and best practices information emerges from the field.

### 1. Log Message Interleaving

By directing the logging output of potentially multiple instances of yangcli-pro and netconfd-pro to syslog, message activity between different instances may be captured and reported in a single chronological history.

## 4.5 Pthreads Images

A NETCONF-PRO image that uses POSIX threads to improve performance on multi-core systems may be built by including **PTHREADS=1** on the make command line. (See the Developers Manual for more information on this option.)

In order to identify individual threads in log output, a thread/session identifier is added to the pre-pended header information, when **--log-header="custom"** is used as a CLI or config parameter when the server is run.

This thread/session id has the form:

**[tid.sid]** where tid is a thread id, and sid is a session id.

Note that thread ids begin with 1 and increment forever, as new threads are created. On the other hand, session ids are limited, occurring within a range of {1 .. max}, where "max" is the maximum number of sessions allowed on the server. (See the **--max-sessions** parameter as described in the Netconfd-Pro manual for more information on this.) A session id of "0" indicates "no session" is associated with the thread in question.

Session ids may be reused as sessions come and go on the server. Note that there is a one-to-one correspondence between threads and sessions, so if the maximum number of sessions is limited, so is the maximum number of threads present on a server at any given time.

Following successful initialization, 4 threads are present on the server. (Note: this may vary between different releases of the server.)

- **[...]**: Main thread
- **[1.0]**: Background thread (Bkgd Thread)
- **[2.0]**: Connect thread (CX Thread)
- **[3.0]**: Timer thread (Tmr Thread)

The "Main" thread represents the initial stream of execution and performs initialization in a way that is essentially identical to that of a non-threaded image.

The background thread has specialized duties that will be described elsewhere in this manual set.

The timer thread is responsible for periodically servicing a timer event queue.

The connect thread listens to a well known socket for new connection requests from clients. It is responsible for creating new sessions. When this happens, a "Receive Thread" (RX Thread) is also created and associated with each new session.

Assume for example that two new client sessions are established soon after the server initializes. Log output headers for these sessions will include one of the following thread/session ids:

- **[4.1]** = Thread 4, session 1
- **[5.2]** = Thread 5, session 2

Note that one should never see the same session id with two different thread ids, unless the old session was closed or killed (see **close-session** and **kill-session** commands in the Yangcli-Pro manual) and a *new* session created that reused the old session id.

## 4.6 Bootstrap CLI

Since YumaPro 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-xxx**: All the logging parameters described in the previous section.
- **--modpath**: use the specified module search path. This will override the `$YUMAPRO_MODPATH` environment variable, if it is set
- **--yumapro-home**: use the specified project root. This will override the `$YUMAPRO_HOME` environment variable, if it is set

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

## 4.7 Configuration Parameters

Command line parameters are used to provide input to YumaPro programs when they are invoked. They are also used extensively by the **yangcli-pro** 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.

### 4.7.1 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
```

## 4.7.2 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-pro {
    ncx:cli;

    // all the yangcli-pro CLI parameters
}
```

If this extension is present, then **netconfd-pro** 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.

## 4.7.3 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-pro {
    ncx:cli;
    ncx:default-parm module;

    // all the yangdump-pro CLI parameters
}
```

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

```
mydir> yangdump-pro --module=test1 --module=test2
mydir> yangdump-pro 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.

```
mydir> yangdump-pro 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.

```
mydir> yangdump-pro *.yang  
mydir> yangdump-pro --subtree=.
```

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

## 5 XPath Reference

The XPath 1.0 path specification language is supported in all YumaPro 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-pro** 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-pro** and **netconfd-pro** 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-pro> if "module-loaded('yuma-system', '2009-12-27')"  
yangcli-pro>   log-info 'correct yuma-system module loaded'  
yangcli-pro> else  
yangcli-pro>   log-error 'Wrong yuma-system module loaded'  
yangcli-pro> 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-pro> if "feature-enabled('mymodule', 'myfeature')"  
yangcli-pro>   log-info 'myfeature is enabled'  
yangcli-pro> else  
yangcli-pro>   log-error 'myfeature is not enabled'  
yangcli-pro> end
```

## 6 Error Reference

All YumaPro 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
0	no error
1	EOF
2 - 99	internal program errors
100 to 199	system errors
200 to 999	user errors
1000 to 1999	warnings
2000 to 2999	informational messages

### 6.1 Error Messages

The current list of error numbers and default error messages can be obtained with the `yangdump-pro` 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-pro 19.10-12 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
14   internal read-write lock error
15   internal mutex error
16   internal heap check error
17   internal spin lock 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 or unexpected attribute  
263 missing element  
264 bad element  
265 unknown or unexpected 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  
381 multiple matches found  
382 no schema default for this node  
383 expected key leaf in list  
384 top-level mandatory objects are not allowed  
385 unknown resource  
386 unended config line block  
387 not supported in evaluation version  
388 unknown resource instance  
389 input data not expected  
390 method not allowed  
391 query parameter not allowed for method  
392 edit pre-condition failed  
393 header not allowed  
394 running config has validation errors  
395 binary file found instead of text file  
396 module is imported by other modules  
397 restricted module cannot be unloaded  
398 request limit reached in evaluation version  
399 IO select call failed  
400 session dropped  
401 media type not in range  
402 an appropriate representation could not be found  
403 data is not in a format acceptable for processing  
404 unknown query parameter  
405 missing Accept header  
406 password is too short  
407 missing input data  
1000 duplicate source  
1001 include file not found  
1002 invalid command line value  
1003 invalid command line option  
1004 command line option unknown  
1005 invalid command line syntax  
1006 missing command line value  
1007 invalid form input  
1008 invalid form  
1009 no instance found  
1010 session closed by remote peer  
1011 duplicate import  
1012 duplicate import with different prefix value  
1013 local typedef not used  
1014 local grouping not used  
1015 import not used  
1016 duplicate unique-stmt argument  
1017 statement ignored  
1018 duplicate include  
1019 include not used  
1020 revision date before 1970

1021 revision date in the future  
1022 enum value order  
1023 bit position order  
1024 invalid status for child node  
1025 duplicate sibling node name from external augment  
1026 duplicate if-feature statement  
1027 using deprecated definition  
1028 XPath object predicate check limit reached  
1029 empty XPath result in must or when expr  
1030 no ancestor node available  
1031 no parent node available  
1032 no child node available  
1033 no descendant node available  
1034 no nodes available  
1035 bad revision-stmt order  
1036 duplicate prefix  
1037 identifier length exceeded  
1038 display line length exceeded  
1039 received unknown capability  
1040 invalid module capability URI  
1041 unknown child node, using anyxml  
1042 invalid value used for parm  
1043 changing object type to string  
1044 using a reserved name  
1045 conf file parm already exists  
1046 no valid revision statements found  
1047 dependency file has errors  
1048 top-level object is mandatory  
1049 file name does not match [sub]module name  
1050 unique-stmt component conditions do not match parent list  
1051 reentrant call detected (retry)