Yuma® Developer Manual

YANG-Based Unified Modular Automation

Table Of Contents

Yuma Developer Manual

1 Preface

1.1 Legal Statements

1.2 Restricted Rights Legend

1.3 Additional Resources

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

Yuma® Installation and Quickstart Guide

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

Yuma® User Manual

To obtain additional support you may email Netconf Central at the e-mail address

support@netconfcentral.com

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

The following section lists the resources available at this time.

1.3.1 WEB Sites

- **Netconf Central**
  - http://www.netconfcentral.org
  - Free information on NETCONF and YANG, tutorials, on-line YANG module validation and documentation database
- **Yang Central**
  - http://www.yang-central.org
  - Free information and tutorials on YANG, free YANG tools for download
- **NETCONF Working Group Wiki Page**

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

1.3.2 Mailing Lists

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

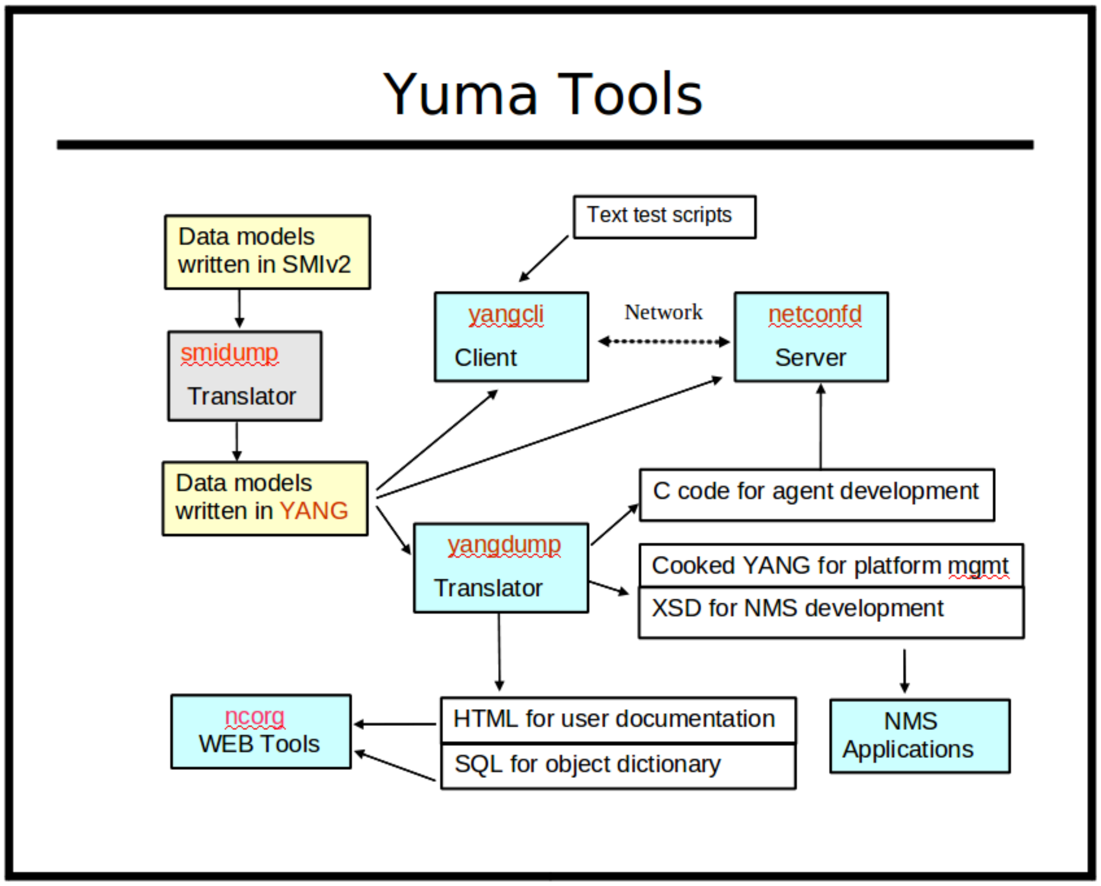
1.4 Conventions Used in this Document

The following formatting conventions are used throughout this document:

Documentation Conventions

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

2 Software Development Overview


Yuma Tools
Text test scripts
Data models written in SMIv2
yangcli Client
Network
netconfd Server
smidump Translator
Data models written in YANG
C code for agent development
yangdump Translator
Cooked YANG for platform mgmt
XSD for NMS development
ncorg WEB Tools
HTML for user documentation
SQL for object dictionary
NMS Applications

2.1 Introduction

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

This section focuses on the software development aspects of NETCONF, YANG, and the **netconfd** server.

2.1.1 What does Yuma Do?

The Yuma Tools suite provides automated support for development and usage of network management information. Refer to the Yuma User Guide for an introduction to the YANG data modeling language and the NETCONF protocol.

This section describes the Yuma development environment and the basic tasks that a software developer needs to perform, in order to integrate YANG module instrumentation into a device.

This manual contains the following information:

- Yuma Development Environment

- Yuma Runtime Environment
- Yuma Source Code Overview
- Yuma Server Instrumentation Library Development Guide

Yuma Tools programs are written in the C programming language, using the 'gnu99' C standard, and should be easily integrated into any operating system or embedded device that supports the Gnu C compiler.

2.1.2 What is a SIL?

A SIL is a Server Instrumentation Library. It contains the 'glue code' that binds YANG content (managed by the **netconfd** server), to your networking device, which implements the specific behavior, as defined by the YANG module statements.

The **netconfd** server handles all aspects of the NETCONF protocol operation, except data model semantics that are contained in description statements. The server uses YANG files directly, loaded at boot-time or run-time, to manage all NETCONF content, operations, and notifications.

Callback functions are used to hook device and data model specific behavior to database objects and RPC operations. The **yangdump** program is used to generate the initialization, cleanup, and 'empty' callback functions for a particular YANG module. The callback functions are then completed (by you), as required by the YANG module semantics. This code is then compiled as a shared library and made available to the **netconfd** server. The 'load' command (via CLI, configuration file, protocol operation) is used (by the operator) to activate the YANG module and its SIL.

2.1.3 Intended Audience

This document is intended for developers of server instrumentation library software, which can be used with the programs in the Yuma suite. It covers the design and operation of the **netconfd** server, and the development of server instrumentation library code, intended for use with the **netconfd** server.

2.1.4 Basic Development Steps

The steps needed to create server instrumentation for use within Yuma are as follows:

- **Create the YANG module data model** definition, or use an existing YANG module.
  - Validate the YANG module with the **yangdump** program and make sure it does not contain any errors. All warnings should also be examined to determine if they indicate data modeling bugs, or not.
  - `Example toaster.yang`
- Make sure the **$$YUMA_HOME** environment variable is defined, and pointing to your Yuma development tree.
- **Create a SIL development subtree**
  - Generate the directory structure and the Makefile with the **yuma_make_sil** script, found in the **netconf/bin** directory in the Yuma development tree.
  - `Example: mydir> $YUMA_HOME/bin/yuma_make_sil toaster`
- **Create the C source code files** for the YANG module callback functions
  - Generate the H and C files with the 'make code' command, in the SIL 'src' directory
  - `Example: mydir/toaster/src> make code`

- **Use your text editor to fill in the device-specific instrumentation** for each object, RPC method, and notification. Almost all possible NETCONF-specific code is either handled in the central stack, or generated automatically. so this code is responsible for implementing the semantics of the YANG data model.
- **Compile your code**
  - Use the 'make' command in the SIL 'src' directory. This should generate a library file in the SIL 'lib' directory.
  - `Example:  mydir/toaster/src> make`
- **Install the SIL library so it is available to the netconfd server.**
  - Use the 'make install' command in the SIL 'src' directory.
  - `Example:  mydir/toaster/src> make install`
- **Run the netconfd server** (or build it again if linking with static libraries)
- **Load the new module**
  - Be sure to add a 'load' command to the configuration file if the module should be loaded upon each reboot.
  - `yangcli Example: load toaster`
- The **netconfd** server will load the specified YANG module and the SIL and make it available to all sessions.

2.2 Server Operation

2.2.1 Initialization

The file netconfd.c contains the initial 'main' function that is used to start the server.

2.2.2 Start a Session

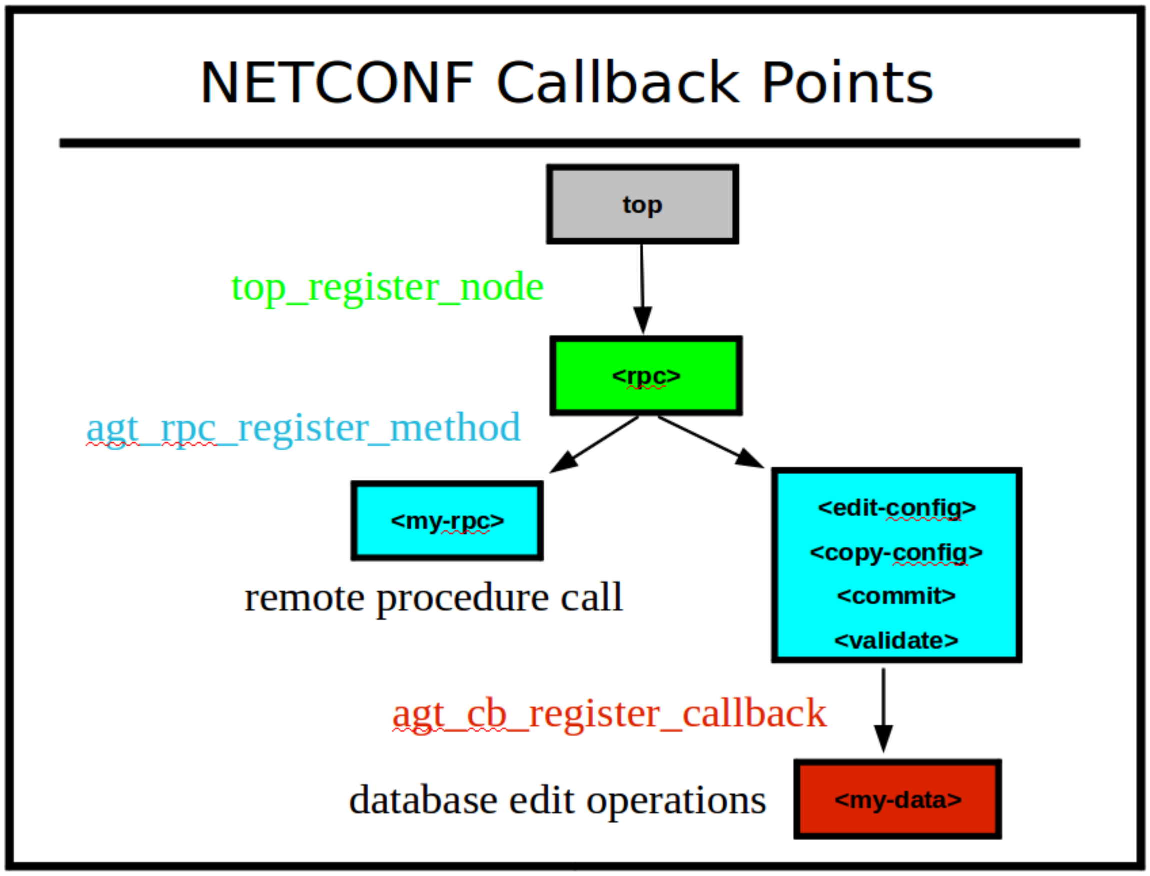
2.2.3 Incoming Requests

2.2.4 Edit the Database

2.2.5 Save the Database

2.2.6 End a Session

2.3 Callback Functions



2.3.1 Initialization and Cleanup

2.3.2 RPC Operations

2.3.3 Database Operations

2.3.4 YANG Extension Data

2.4 Main Data Structures

2.4.1 NCX_MODULE_T

2.4.2 OBJ_TEMPLATE_T

2.4.3 VAL_VALUE_T

2.4.4 SES_CB_T

2.4.5 RPC_MSG_T

2.4.6 XML_NODE_T

2.4.7 STATUS_T

2.5 Toaster Tutorial

3 Development Environment

3.1 Programs and Libraries Needed

There are several components used in the Yuma software development environment:

- gcc compiler and linker
- ldconfig and install programs
- GNU make program
- shell program, such as bash
- Yuma development tree: the source tree containing Yuma Tools code, specified with the **$$YUMA_HOME** environment variable.
- SIL development tree: the source tree containing server instrumentation code

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

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

The following program is part of Yuma Tools, and needs to be installed:

- **netconf-subsystem**
  - The thin client sub-program that is called by **sshd** when a new SSH2 connection to the 'netconf' sub-system is attempted.
    - This program will use an AF_LOCAL socket, using a proprietary **<ncxconnect>** message, to communicate with the **netconfd** server..
    - After establishing a connection with the **netconfd** server, this program simply transfers SSH2 NETCONF channel data between **sshd** and **netconfd**.

The following program is part of Yuma Tools, and usually found within the Yuma development tree:

- **netconfd**
  - The NETCONF server that processes all protocol operations.
    - The **agt_ncxserver** component will listen for **<ncxconnect>** messages on the designated socket (usually /tmp/ncxserver.sock). If an invalid message is received, the connection will be dropped. Otherwise, the **netconf-subsystem** will begin passing NETCONF channel data to the netconfd server. The first message is expected to be a valid <hello> PDU. If

The following external libraries are used by Yuma, and need to be pre-installed:

- **libtecla**
  - command line library provided with Yuma Tools; used by **yangcli**
- **ncurses**
  - character processing library needed by libtecla; used by **yangcli**
- **libc (or glibc)**

  - unix system library
- **libssh2**
  - SSH2 client library, used by yangcli
- **libxml2**
  - xmlTextReader XML parser
  - pattern support

3.2 Yuma Source Tree Layout

3.3 Platform Profile

3.4 SIL Makefile

3.4.1 Build Targets

3.4.2 Command Line Build Options

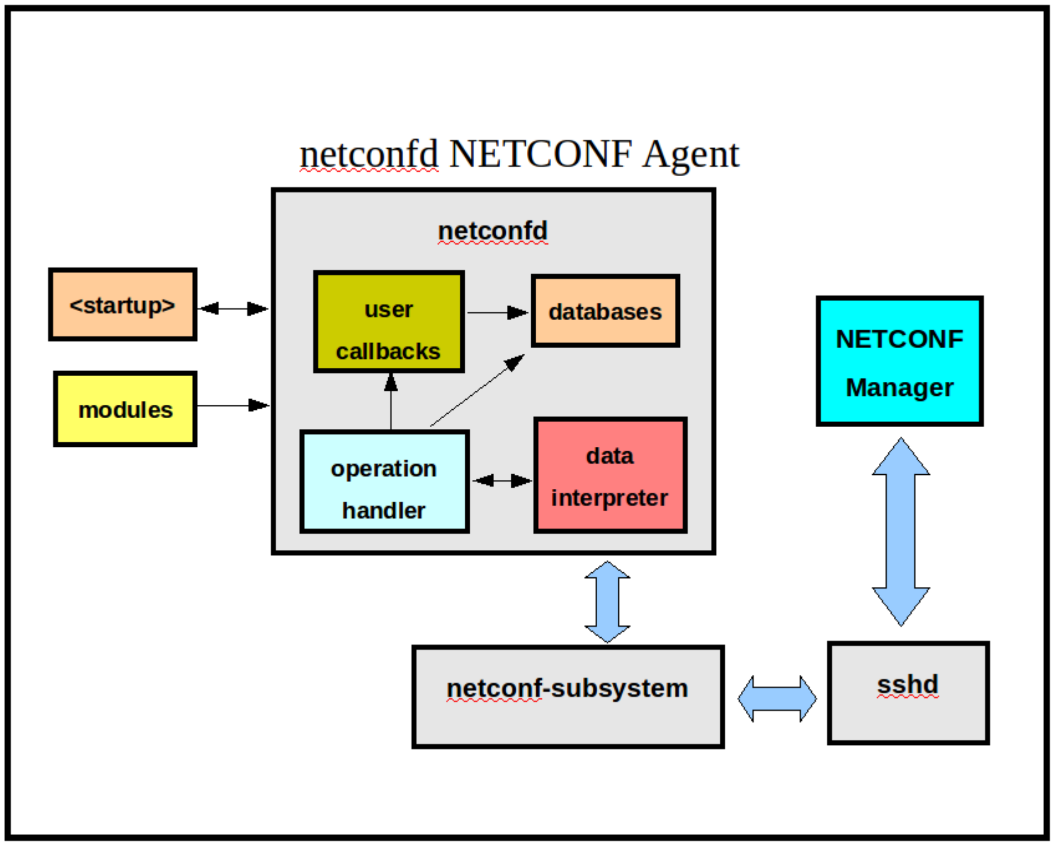
DEBUG

STATIC

MEMTRACE

MAC

4 Runtime Environment



4.1 Memory Management

4.2 Capability Management

4.3 Session Management

4.4 Database Management

4.5 Network Input and Output

4.6 Logging and Debugging

4.7 XML

4.8 XPath

4.9 YANG Data Structures

4.10 NETCONF Operations

4.11 RPC Reply Generation

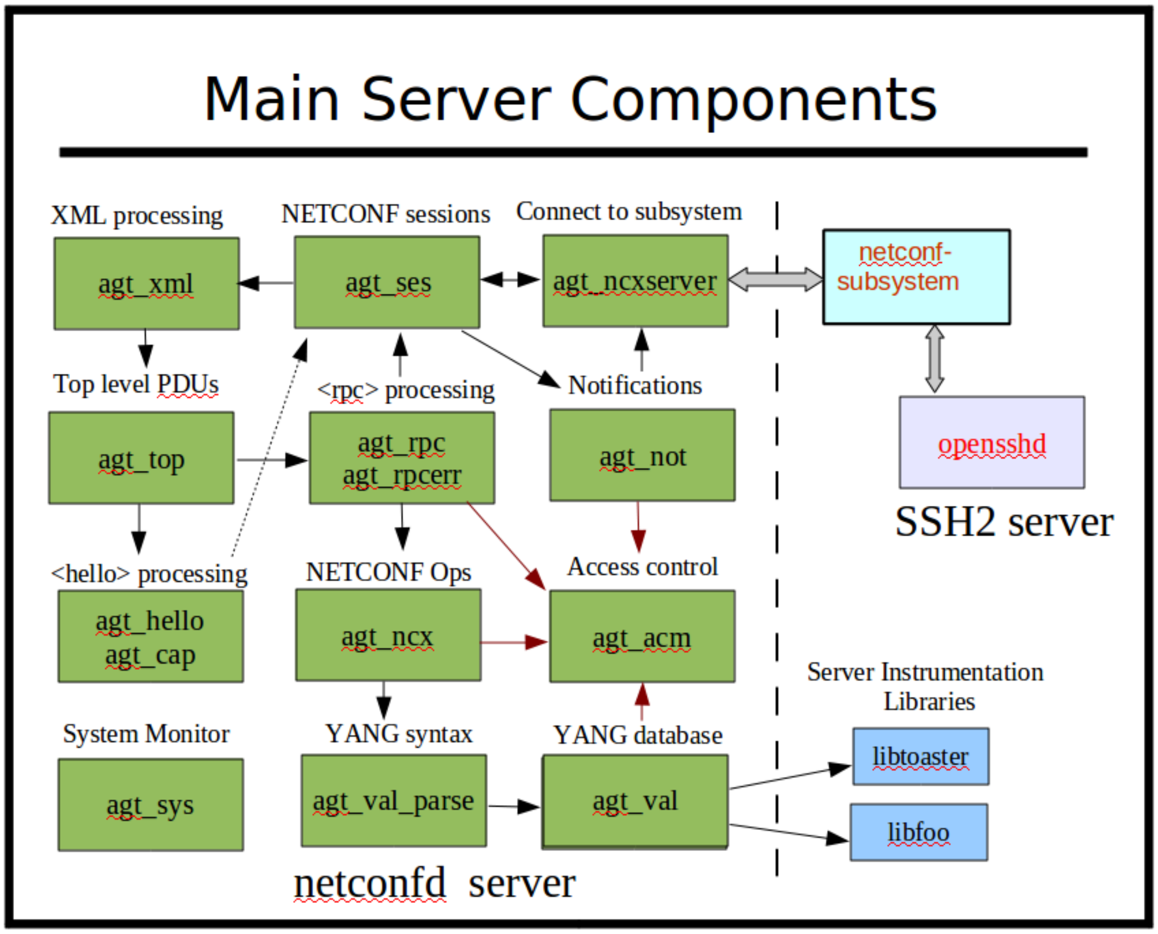
4.12 RPC Error Reporting

4.13 Notifications

4.14 Retrieval Filtering

4.15 Access Control

4.16 Message Flows



The **netconfd** server provides the following type of components:

- NETCONF session management
- NETCONF/YANG database management
- NETCONF/YANG protocol operations
- Access control configuration and enforcement
- RPC error reporting
- Notification subscription management
- Default data retrieval processing
- Database editing
- Database validation
- Subtree and XPath retrieval filtering
- Dynamic and static capability management
- Conditional bbject management (if-feature, when)
- Memory management
- Logging management

- Timer services

All NETCONF and YANG protocol operation details are handled automatically within the **netconfd** server. All database locking and editing is also handled by the server. There are callback functions available at different points of the processing model for your module specific instrumentation code to process each server request, and/or generate notifications. Everything except the 'description statement' semantics are usually handled

The server instrumentation stub files associated with the data model semantics are generated automatically with the **yangdump** program. The developer fills in server callback functions to activate the networking device behavior represented by each YANG data model.

4.17 Automation Control

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

4.17.1 YANG Language Extensions

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

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

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

YANG Language Extensions

extension	description
ncx:hidden;	Declares that the object definition should be hidden from all automatic documentation generation. Help will not be available for the object in **yangcli**.
ncx:metadata “*attr-type attr-name*”;	Defines a qualified XML attribute in the module namespace. Allowed within an RPC input parameter. **attr-type** is a valid type name with optional YANG prefix. **attr-name** is the name of the XML attribute.
ncx:no-duplicates;	Declares that the **ncx:xsdlist** data type is not allowed to contain duplicate values. The default is to allow duplicate token strings within an

	ncx:xsdlist value.
ncx:password;	Declares that a string data type is really a password, and will not be displayed or matched by any filter.
ncx:qname;	Declares that a string data type is really an XML qualified name. XML prefixes will be properly generated by **yangcli** and **netconfd**.
ncx:root;	Declares that the container parameter is really a NETCONF database root, like <config> in the <edit-config> operations. The child nodes of this container are not specified in the YANG file. Instead, they are allowed to contain any top-level object from any YANG file supported by the server.
ncx:schema-instance;	Declares that a string data type is really an special schema instance identifier string. It is the same as an instance-identifier built-in type except the key leaf predicates are optional. For example, missing key values indicate wild cards that will match all values in **nacm** <dataRule> expressions.
ncx:secure;	Declares that the database object is a secure object. If the object is an **rpc** statement, then only the **netconfd** 'superuser' will be allowed to invoke this operation by default. Otherwise, only read access will be allowed to this object by default, Write access will only be allowed by the 'superuser', by default.
ncx:very-secure;	Declares that the database object is a very secure object. Only the 'superuser' will be allowed to access the object, by default.
ncx:xsdlist “*list-type*”;	Declares that a string data type is really an XSD style list. **list-type** is a valid type name with optional YANG prefix. List processing within <edit-config> will be automatically handled by **netconfd**.
ncx:xpath;	Declares that a string data type is really an XPath expression. XML prefixes and all XPath processing will be done automatically by **yangcli** and **netconfd**.

4.17.2 ncx:cli Extension

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

```
container yangcli {
    ncx:cli;

    // all the yangcli CLI parameters
}
```

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

4.17.3 ncx:default-parm Extension

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

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

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

    // all the yangdump CLI parameters
}
```

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

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

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

```
yangdump log-app  test1
```

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

5 Software Components

5.1 ncx

5.1.1 b64.c

5.1.2 BLOB.C

5.1.3 BOBHASH.C

5.1.4 CAP.C

5.1.5 CFG.C

5.1.6 CLI.C

5.1.7 CONF.C

5.1.8 DEF_REG.C

5.1.9 DLQ.C

5.1.10 EXT.C

5.1.11 GRP.C

5.1.12 HELP.C

5.1.13 LOG.C

5.1.14 NCX.C

5.1.15 NCXMOD.C

5.1.16 OBJ.C

5.1.17 OBJ_HELP.C

5.1.18 OP.C

5.1.19 RPC.C

5.1.20 RPC_ERR.C

5.1.21 RUNSTACK.C

5.1.22 SEND_BUFF.C

5.1.23 SES.C

5.1.24 SES_MSG.C

5.1.25 STATUS.C

5.1.26 TK.C

5.1.27 TOP.C

5.1.28 TSTAMP.C

5.1.29 TYP.C

5.1.30 VAL.C

5.1.31 VAL_UTIL.C

5.1.32 VAR.C

5.1.33 XML_MSG.C

5.1.34 XMLNS.C

5.1.35 XML_UTIL.C

5.1.36 XML_VAL.C

5.1.37 XML_WR.C

5.1.38 XPATH1.C

5.1.39 XPATH.C

5.1.40 XPATH_WR.C

5.1.41 XPATH_YANG.C

5.1.42 YANG.C

5.1.43 YANG_EXT.C

5.1.44 YANG_GRP.C

5.1.45 YANG_OBJ.C

5.1.46 YANG_PARSE.C

5.1.47 YANG_TYP.C

5.2 agt

5.2.1 AGT_ACM.C

5.2.2 AGT.C

5.2.3 AGT_CAP.C

5.2.4 AGT_CB.C

5.2.5 AGT_CLI.C

5.2.6 AGT_CONNECT.C

5.2.7 AGT_EXPR.C

5.2.8 AGT_HELLO.C

5.2.9 AGT_IF.C

5.2.10 AGT_NCX.C

5.2.11 AGT_NCXSERVER.C

5.2.12 AGT_NOT.C

5.2.13 AGT_PROC.C

5.2.14 AGT_RPC.C

5.2.15 AGT_RPCERR.C

5.2.16 AGT_SES.C

5.2.17 AGT_SIGNAL.C

5.2.18 AGT_STATE.C

5.2.19 AGT_SYS.C

5.2.20 AGT_TIMER.C

5.2.21 AGT_TOP.C

5.2.22 AGT_TREE.C

5.2.23 AGT_UTIL.C

5.2.24 AGT_VAL.C

5.2.25 AGT_VAL_PARSE.C

5.2.26 AGT_XML.C

5.2.27 AGT_XPATH.C

5.3 mgr

5.3.1 MGR.C

5.3.2 MGR_CAP.C

5.3.3 MGR_HELLO.C

5.3.4 MGR_IO.C

5.3.5 MGR_NOT.C

5.3.6 MGR_RPC.C

5.3.7 MGR_SES.C

5.3.8 MGR_SIGNAL.C

5.3.9 MGR_TOP.C

5.3.10 MGR_VAL_PARSE.C

5.3.11 MGR_XML.C

5.4 platform

5.4.1 CURVERSION.H

5.4.2 PLATFORM.PROFILE

5.4.3 PLATFORM.PROFILE.DEPEND

5.4.4 PLATFORM.PROFILE.SIL

5.4.5 PROCDEFS.H

5.4.6 SETVERSION.SH

5.5 subsys

5.5.1 NETCONF-SUBSYSTEM.C

5.6 netconfd

5.6.1 NETCONFD.C

5.7 yangcli

5.7.1 YANGCLI_AUTOLOAD.C

5.7.2 YANGCLI_AUTOLOCK.C

5.7.3 YANGCLI.C

5.7.4 YANGCLI_CMD.C

5.7.5 YANGCLI_LIST.C

5.7.6 YANGCLI_SAVE.C

5.7.7 YANGCLI_SHOW.C

5.7.8 YANGCLI_TAB.C

5.7.9 YANGCLI_UTIL.C

5.8 yangdump

5.8.1 C.C

5.8.2 C_UTIL.C

5.8.3 CYANG.C

5.8.4 H.C

5.8.5 HTML.C

5.8.6 SQL.C

5.8.7 XSD.C

5.8.8 XSD_TYP.C

5.8.9 XSD_UTIL.C

5.8.10 XSD_YANG.C

5.8.11 YANGDUMP.C

5.8.12 YANGDUMP_UTIL.C

5.9 yangdiff

5.9.1 YANGDIFF.C

5.9.2 YANGDIFF_GRP.C

5.9.3 YANGDIFF_OBJ.C

5.9.4 YANGDIFF_TYP.C

5.9.5 YANGDIFF_UTIL.C