Network Configuration with XML

NCX Overview

Andy Bierman
April 4, 2009
Netconf Central

andy@netconfcentral.com

If your only tool is a hammer...

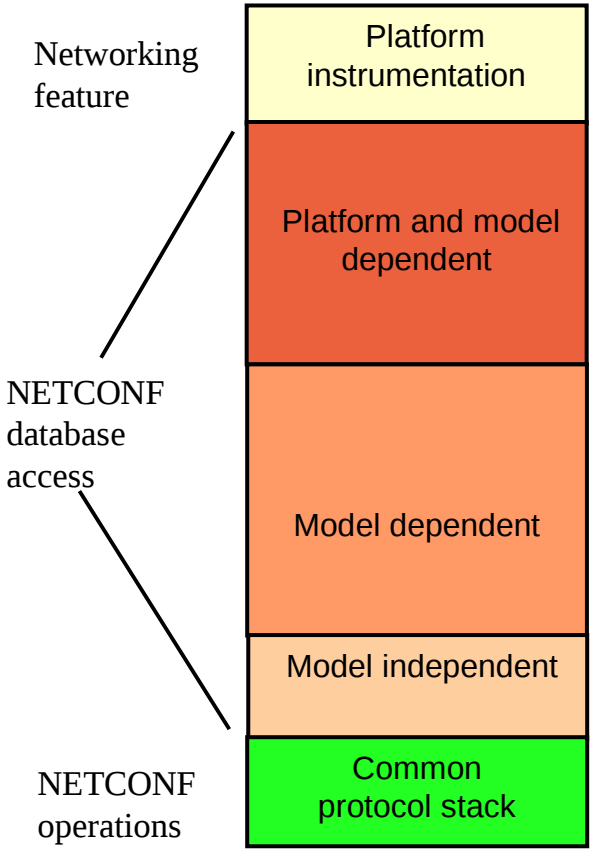
Your NM Tools

Your Tools with
NETCONF/YANG

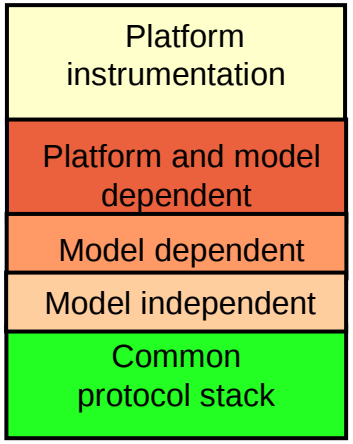
Contents

- Introduction to NCX and YANG
- NCX System Components
- NCX Implementation

NETCONF Development Costs



Without automation

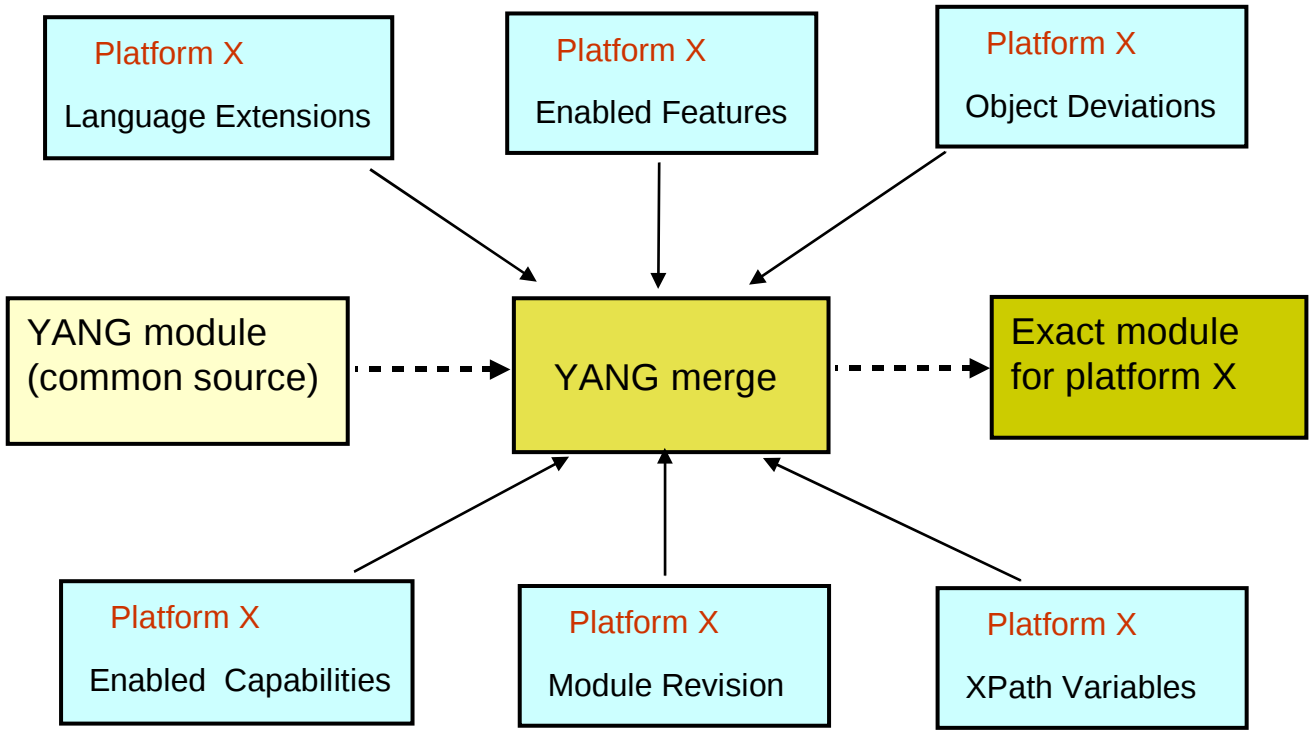


With YANG automation

What is NCX?

- Network Configuration Management Toolkit
  - » Completely Standards-based
    - – NETCONF and SSH2 protocols
    - – YANG data modeling language
    - – XML encoding and Xpath filtering
    - – ietf-netconf-state monitoring data model
    - – ietf-netconf-partial-lock database locking extensions
  - » Full automation of NETCONF protocol handling
    - – All session, operation, database access, error, and XML handling done in the toolkit or in the agent central stack
    - – Utilization of YANG extensions, deviations, and other automation support features

YANG Cooked Modules

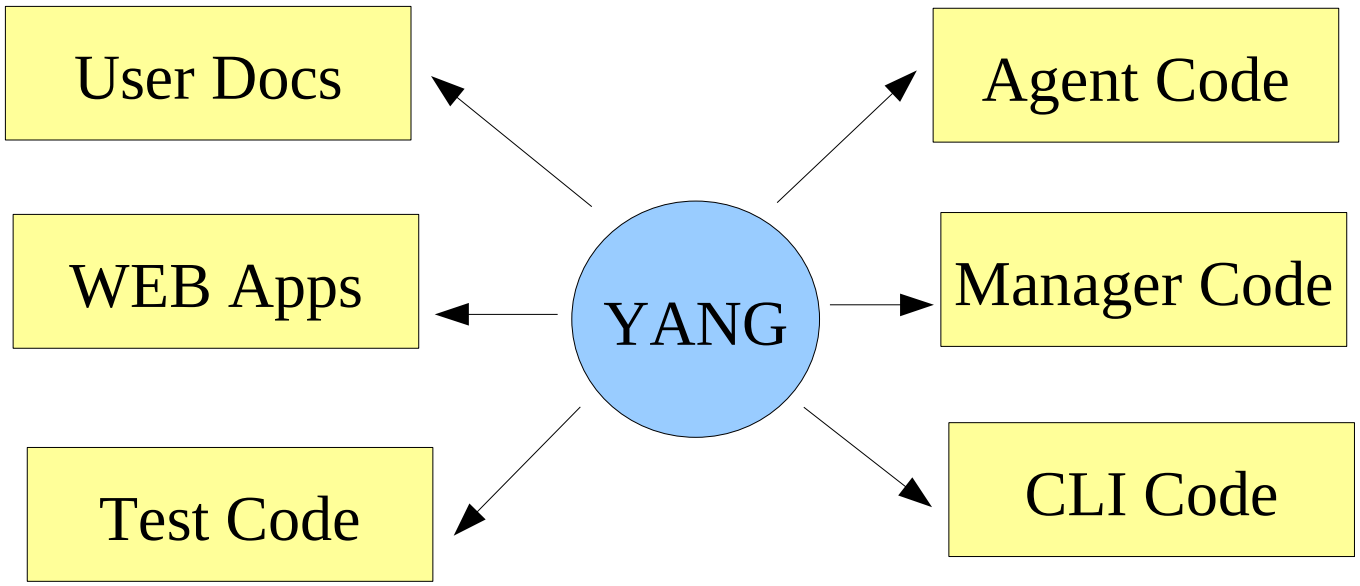


Platform-specific schema created automatically

What is YANG?

- Yet Another Next Generation Data Modeling Language for the NETCONF Protocol
  - » YANG will become the SMIv2 replacement for data model module definition within the Internet community
  - » Arbitrarily complex data structures are supported, making it possible for data model designers to design the MIB first, not last, saving lots of time
  - » YANG is designed from the ground up to support tool automation

YANG as Source Code



YANG-based Automation

- YANG allows clever tools to automate the development and run-time support for the entire NETCONF protocol, by using machine-readable text instead of DESCRIPTION clauses:
  - » feature/if-feature
  - » must/when statements
  - » grouping/uses/refine statements
  - » object deviations
  - » language extensions

YANG Module Partitions

- YANG features
  - » A feature defines a conceptual partition within a module
  - » Objects with the matching if-feature statement(s) are part of the logical partition
  - » Supported features are static, set at code compile-time and advertised by the agent at session start-up time
  - » A code generator can remove unneeded code for the unimplemented features on each platform

YANG Conditionals

- must and when statements
  - » A must statement defines inter-object value-space conditions, using an XPath expression
    - – Each data object can have zero or more must statements
    - – All the expressions must be 'true' for the object that contains the must statement(s) to be considered valid
  - » A when statement defines inter-object existence conditions, using an Xpath expression
    - – Each object can have zero or one when-statement
    - – The expression must be 'true' for the object that contains the when statement to be considered present in the agent
    - – Objects with 'false' when expression values are automatically deleted by the agent

YANG Object Reuse

- grouping, uses, and refine statements
  - » A grouping statement defines an abstract container of reusable object definitions
    - – Each grouping can contain its own types and groupings, and extend/refine other groupings via 'uses'
  - » A uses statement defines an instantiation of a particular grouping (real objects)
    - – Each usage can be tailored to exact needs, using mechanisms built into YANG
  - » A refine statement 'customizes' an object used from a grouping
    - – Code generators can utilize this feature as well as humans

Platform-specific Deviations

- deviation and deviate statements
  - » A deviation statement defines the alterations to one data model object for an agent implementation
    - – Super-charged SNMP AGENT-CAPABILITIES
    - – Built-in 'patch' operation for MIB objects
    - – Contains 1 or more 'deviate' statements
  - » A deviate statement defines the patch operation
    - – *not-supported* operation removes an object completely
    - – *add*, *replace*, and *delete* operations manipulate the sub-clauses within an object definition
  - » Tools can automatically generate data model definitions that exactly match each agent platform

Language Extensions

- extension statement
  - » An extension statement is an annotation that defines the syntax for a tool-specific YANG language extension
    - – All YANG tools must handle any extension usage gracefully, not just their own extension definitions
    - – Similar to compiler directives in C code or processing instructions in XML
  - » Any complex program behavior can be automated by annotating the YANG module definitions with the right extensions
  - » Some extensions from ncx.yang:
    - – root, cli, default-parm
    - – secure, very-secure, xpath, hidden

Some NCX Extensions (1)

- ncx:root;
  - » Declares an empty container to be a database root, so it can be automatically processed and not hard-wired into RPC operations like <get>, <edit-config> ,etc.
- ncx:cli;
  - » Declares a YANG container definition to be used as the command line and configuration file interface for any program, instead of NETCONF content
    - – All documentation, compile-time, and run-time CLI code can be automatically generated with 1 extra line of YANG code
- ncx:default-parm “foo”;
  - » Defines a default parameter name for CLI or RPC input if it is omitted by the user (used by manager applications)

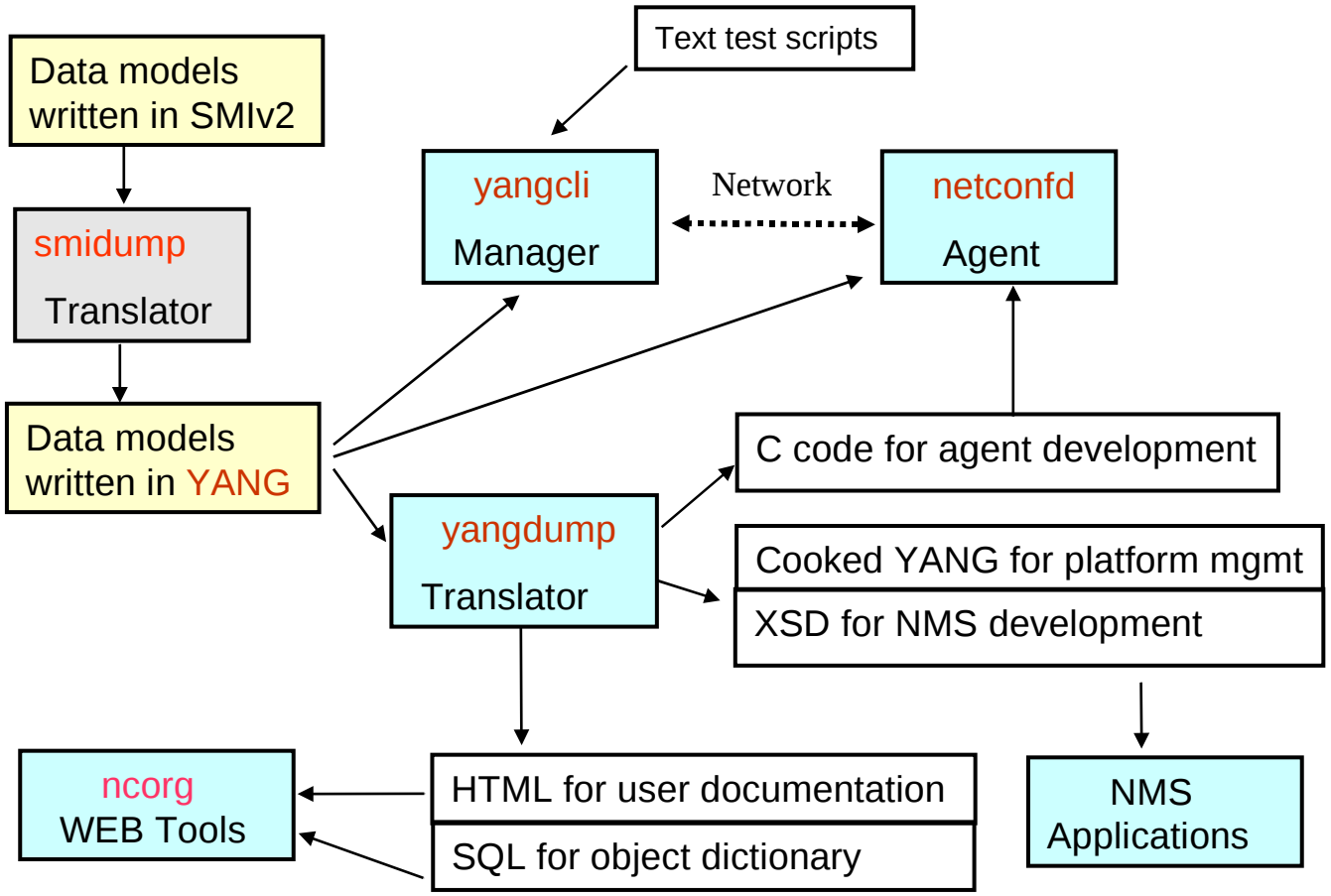
Some NCX Extensions (2)

- ncx:secure;
  - » Declares an object to be security sensitive
    - – Only 'root' write access is allowed, even if no ACM
- ncx:very-secure;
  - » Declares an object to be very security sensitive
    - – Only 'root' access is allowed, even if no ACM
- ncx:xpath;
  - » Declares an object or data type to be an Xpath expression
    - – tools can automate XML namespace and validation code
- ncx:hidden;
  - » Removes an object from documentation and help text

Summary: Part 1

- The YANG module becomes the source code, replacing the need to manually implement almost any data model programming task
- Automation tools can be developed which remove 50% to 90% of the development costs related to network management software development
- 2nd and 3rd party application/feature development for complex networking devices is now possible

Part 2: System Components



smidump

- SMIv2 translation to YANG
  - » Part of the well-established libsmi toolset for SNMP
  - » 'smidump -f yang' converts SMIv2 to YANG
  - » A direct automated translation is needed to allow NETCONF operations (and filtering) to be applied to SMIv2-based agent instrumentation
  - » YANG could become SMIv3 in the IETF
    - – Several people have already stated in IETF meetings they want to use YANG, not SMIv2 from now on
    - – If all SMIv2 modules can be automatically translated with a free tool, then there is no reason to continue using SMIv2

yangdump

- Validation of YANG source files
  - » Supports all YANG constructs
  - » Full Xpath 1.0 implementation
  - » Extensive error and warning reports
  - » Full support for sub-module unification
- Translation to many output formats
  - » HTML for hyper-licked WEB docs
  - » XSD for application data model validation
  - » C code for agent instrumentation
  - » Canonical YANG for documentation and platform-specific representation of YANG files
  - » SQL for WEB application support

ncorg

- TurboGears 1 application providing the Netconf Central WEB site
  - » Uses files generated from yangdump
  - » Provides linked-docs, search, list, browse functionality for any YANG database contents
  - » Provides online validate and diff services
  - » Provides documentation and tutorial information
  - » Could be platform for additional WEB applications
    - – Build a YANG module
    - – User workbench front-end
    - – WEB NETCONF Client application
    - – Integration with bug-tracking, subversion
    - – RSS feed/mailer support for module updates to database

yangcli (1)

- NETCONF over SSH manager tool
  - » Full NETCONF and YANG support
  - » Completely data-driven from YANG files
  - » Context-sensitive help and commands
    - – Based on the capabilities reported for each session
  - » Full Xpath 1.0 support
  - » Complex command line support via libtecla, not readline
  - » Simple script support
  - » Extensive user variable support
    - – System and session parameters available to scripts
    - – Output from any RPC operation can be stored in a user variable
      - $foo = xget /interfaces/interface[name='eth0']

yangcli (2)

- NETCONF over SSH manager tool
  - » Simple-to-use commands for common NETCONF operations:
    - 'create', 'merge', 'replace', delete', 'insert':
      Simplified <edit-config> operations
    - 'save': agent-content specific save-changes operation
    - 'sget' and 'sget-config':
      Simplied <get> and <get-config> with subtree filtering
    - 'xget' and 'xget-config':
      Simplied <get> and <get-config> with XPath filtering
  - » Direct access to arbitrary YANG content
    - 'mgrload foo' is all that is needed to access all the definitions from a new module
      - Any RPC operation can be used as a CLI command

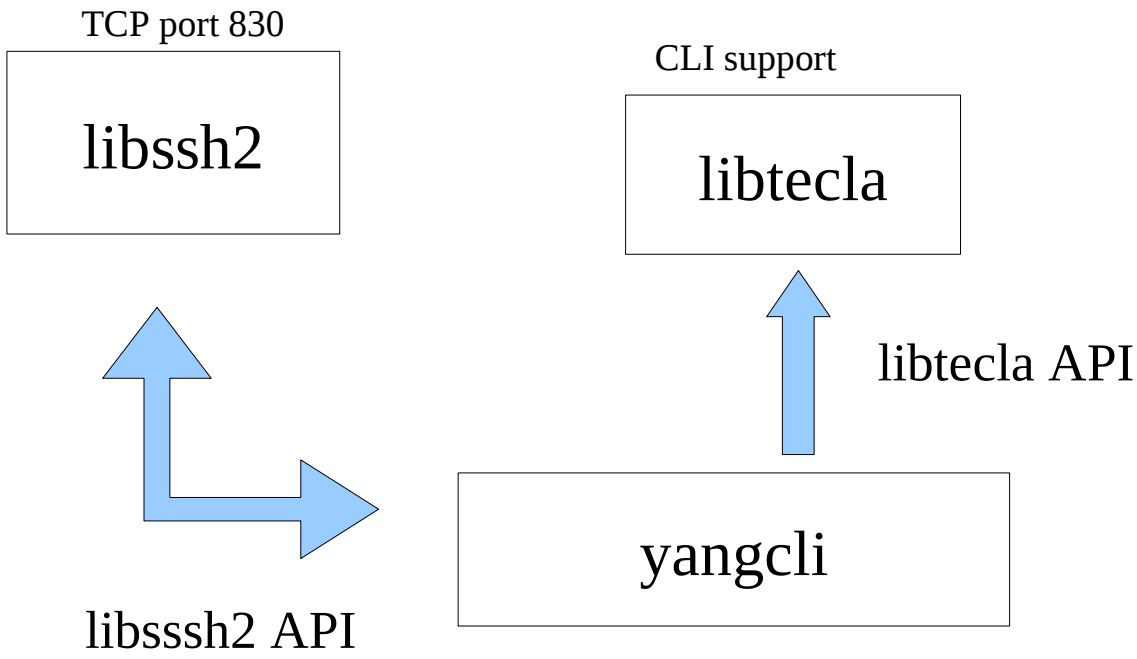
yangcli (3)

- Direct CLI access to YANG RPC commands
  - » RPC method name is the command name and RPC input nodes are the CLI parameters
    - – Plain parameters:
      - kill-session session-id=203
    - – User variables
      - validate config=$testconfig
    - – File parameters:
      - edit-config target=candidate config=@config.xml \ default-operation=continue-on-error
    - – Inline XML parameters:
      - get filter=[<nc:filter>....</nc:filter>]

yangcli (4)

- Direct CLI access to YANG database contents
  - » Simple interactive commands like 'create' and 'merge' use a YANG schema identifier or instance-identifier to select nodes within a database for modification
    - – merge target=/interfaces/interface[name='eth0]/ifMtu
  - » Special RPC methods 'sget' and 'sget-config' convert YANG schema identifier or instance-identifier expressions directly into NETCONF sub-tree filters
  - » XML details such as 'nc:operation' attribute are hidden with specialized editing commands
  - » Many configuration options are provided which allows yangcli to be used as a configuration tool or a software development tool

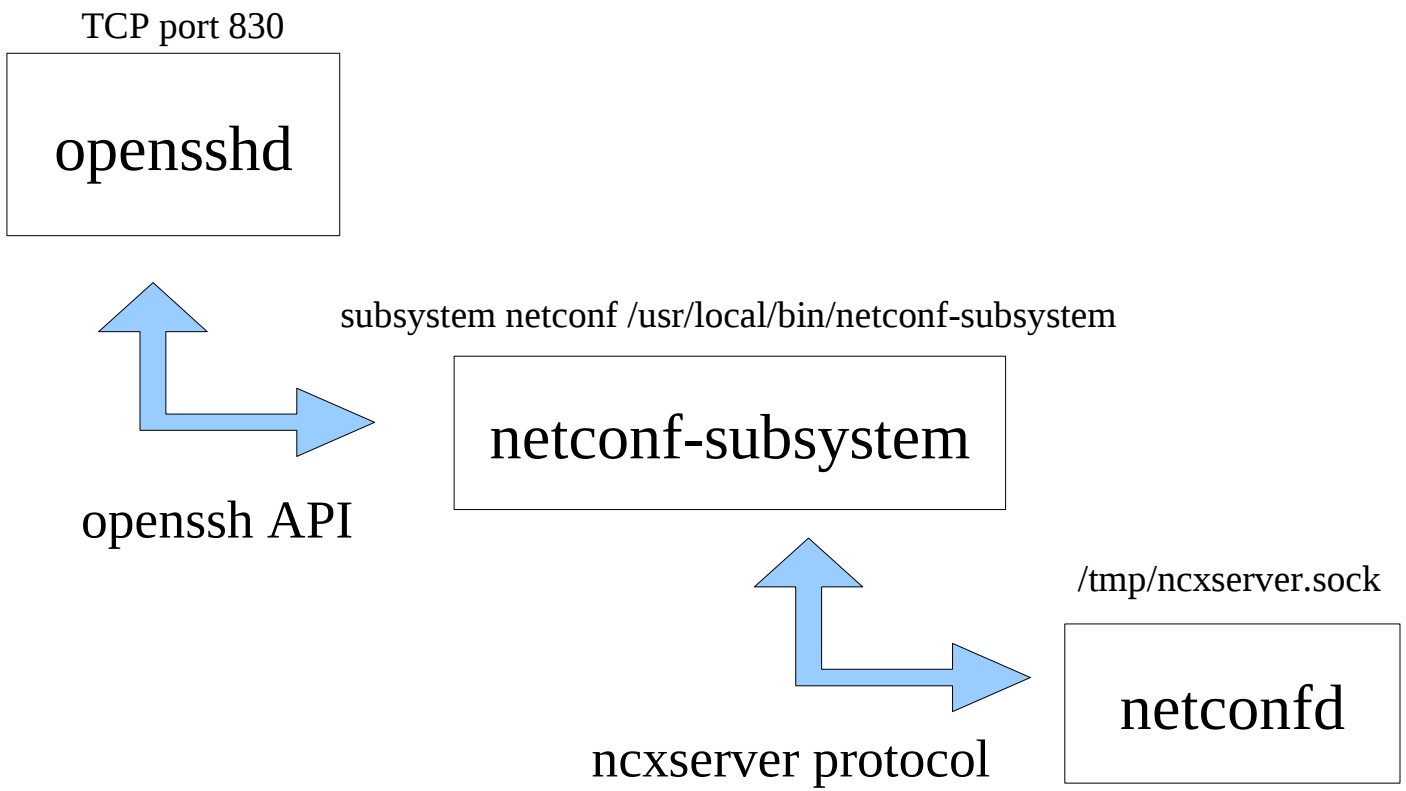
yangcli components



netconfd

- NETCONF over SSH agent
  - » Full implementation of NETCONF except:
    - – :confirmed-commit is not done
    - – :url is not done yet
  - » Native YANG implementation, data-driven directly from YANG source files
  - » Full implementation of all YANG-specific operations and error responses
  - » Extensive YANG-driven conf-file/CLI for customization
  - » Data models implemented:
    - – ietf-netconf-state
    - – ietf-partial-lock
    - – ietf-with-defaults
    - – nacm.yang (NETCONF Access Control Model)

netconfd components



netconfd Automation

- 100% handled in the central stack:
  - » All session, capability, XML, and YANG mechanisms
  - » All validation of every constraint available in YANG
  - » All validation of NCX extensions
  - » All aspects of the <edit-config> operation
  - » All YANG extensions, such as 'insert' and 'key'
  - » All validation and invocation of all NETCONF operations
  - » All streamed subtree and Xpath filtered output
  - » All <rpc-error> collection and reporting
  - » All <candidate/> database <commit> validation
  - » <candidate/> emulation for <running/> as target
  - » All 'false when statement' removed during commit

YANG-o-mation

- YANG is the engine that drives the bus
  - » The standard YANG source file is also the implementation source file
  - » There is no code to write
  - » The YANG file is the code
- Why automation is critical
  - » Lowers development costs by 50% to 90%
  - » Maintain consistency across platforms and releases
  - » Memory trade-off not significant because data-driven code replaces a lot of static code
  - » Domain-specific experts can focus on writing a good data model, and not deal with 'protocol weirdness'

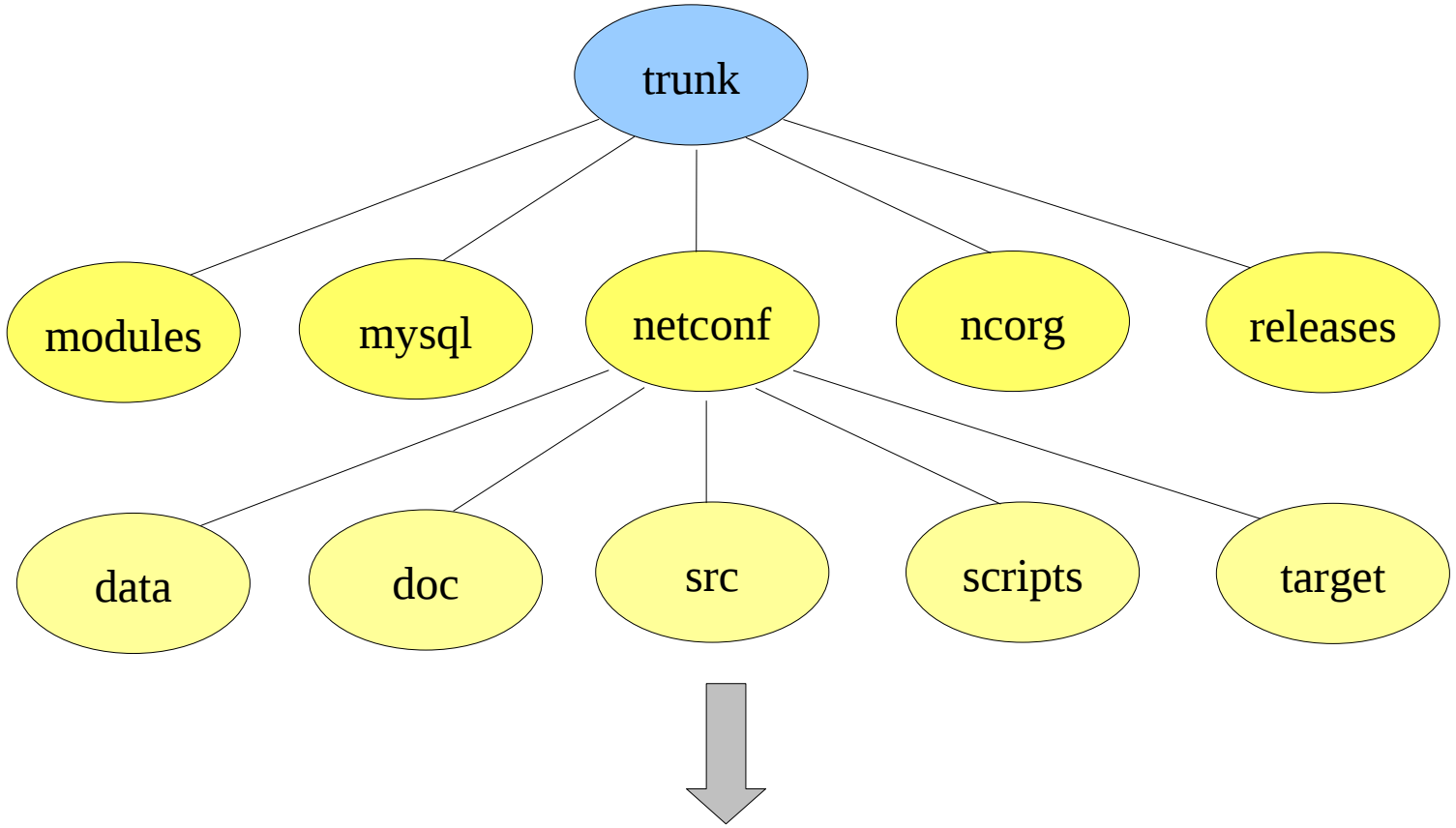
Summary: Part 2

- The NCX tool-set includes tools to automate most aspects of NETCONF application and agent software development
- An extensible layered design allows any significant component to be replaced, for easier platform integration
- 100% standards-based solution adopted by the IETF insures multi-vendor support and the investment in data model definition modules

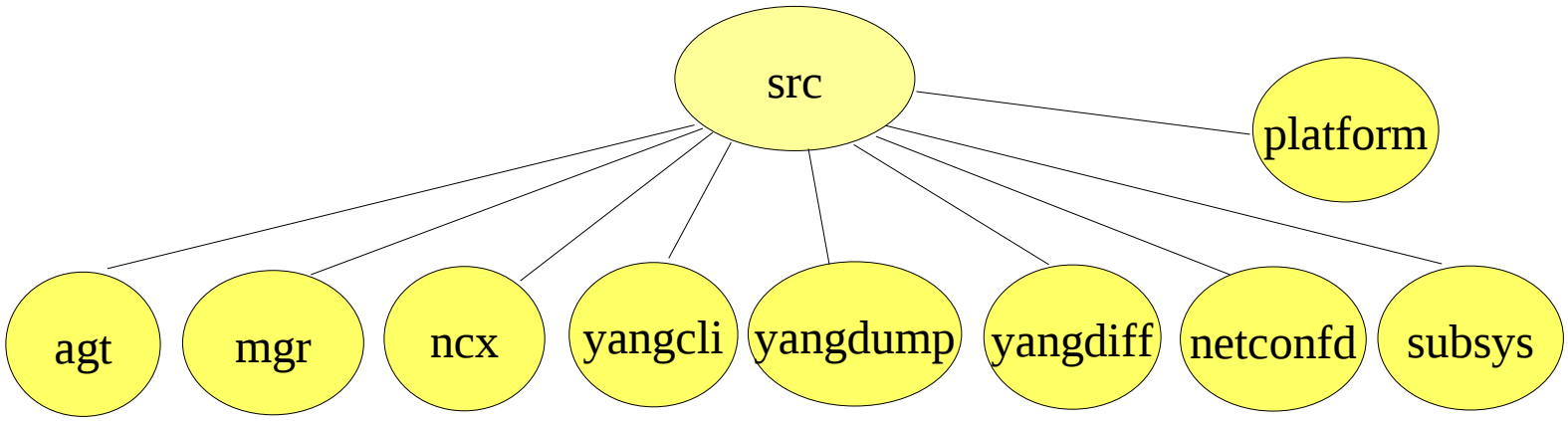
Part 3: NCX Implementation

- Approx. 180,000 lines ISO standard C code
  - » Compiled as 'std=gnu99' to support XPath floating point operations
  - » Supported platforms at this time
    - – Fedora, MacOSX, Ubuntu, OpenSuse
    - – Should compile on any *nix platform, but automake support is not done yet (uses proprietary procdefs.h file now)
- Approx 10,000 lines of Python and KID templates
  - » Used only in the ncorg WEB site code
- All external libraries used have acceptable licensing (e.g., BSD or MIT)
  - » glibc, libtecla, curses, libssh2, libxml2

YangTools subversion tree



C Source Tree



C Source Directories

- agt: NETCONF agent library
- mgr: NETCONF manager library
- ncx: Core common library
- yangcli: yangcli program
- yangdump: yangdump program
- yangdiff: yangdiff program
- netconfd: netconfd program
- subsys: netconf-subsystem program
- platform: platform.profile and procdefs.h

C Coding Conventions

- Super strict coding conventions
- Self-compiling H files
- No complex C coding constructs
- Braces used in every block
- Every function fully commented
- Very few #ifdefs in the actual code (DEBUG)

GNU Compiler Flags

- Super strict compiler checks used:

```
CWARN=-Wall -Wno-long-long -Wformat-y2k -Winit-self \
-Wmissing-include-dirs -Wswitch-default -Wunused-parameter \
-Wextra -Wundef -Wshadow -Wpointer-arith \
-Wwrite-strings -Wbad-function-cast -Wcast-qual -Wcast-align \
-Waggregate-return -Wstrict-prototypes -Wold-style-definition \
-Wmissing-prototypes -Wmissing-declarations \
-Wpacked -Wunreachable-code -Winvalid-pch \
-Wredundant-decls -Wnested-externs -Winline -std=gnu99 -Werror
```

About the Author (1)

- Computer Science degree (SFSU 1987)
- Over 20 years experience and almost 2 million lines of code in commercial embedded network products
  - » DAVID Systems (5 years)
    Operating systems, SNMP stack and agent
  - » SynOptics/Bay Networks (3 years)
    Designed and implemented RMON agent, portable agent architecture
  - » Cisco Systems (10 years)
    Designed and implemented RMON blades, Entity MIB, many other SNMP agent components in IOS and CatOS
  - » Netconf Central (4 years)
    Founded, designed and implemented YangTools

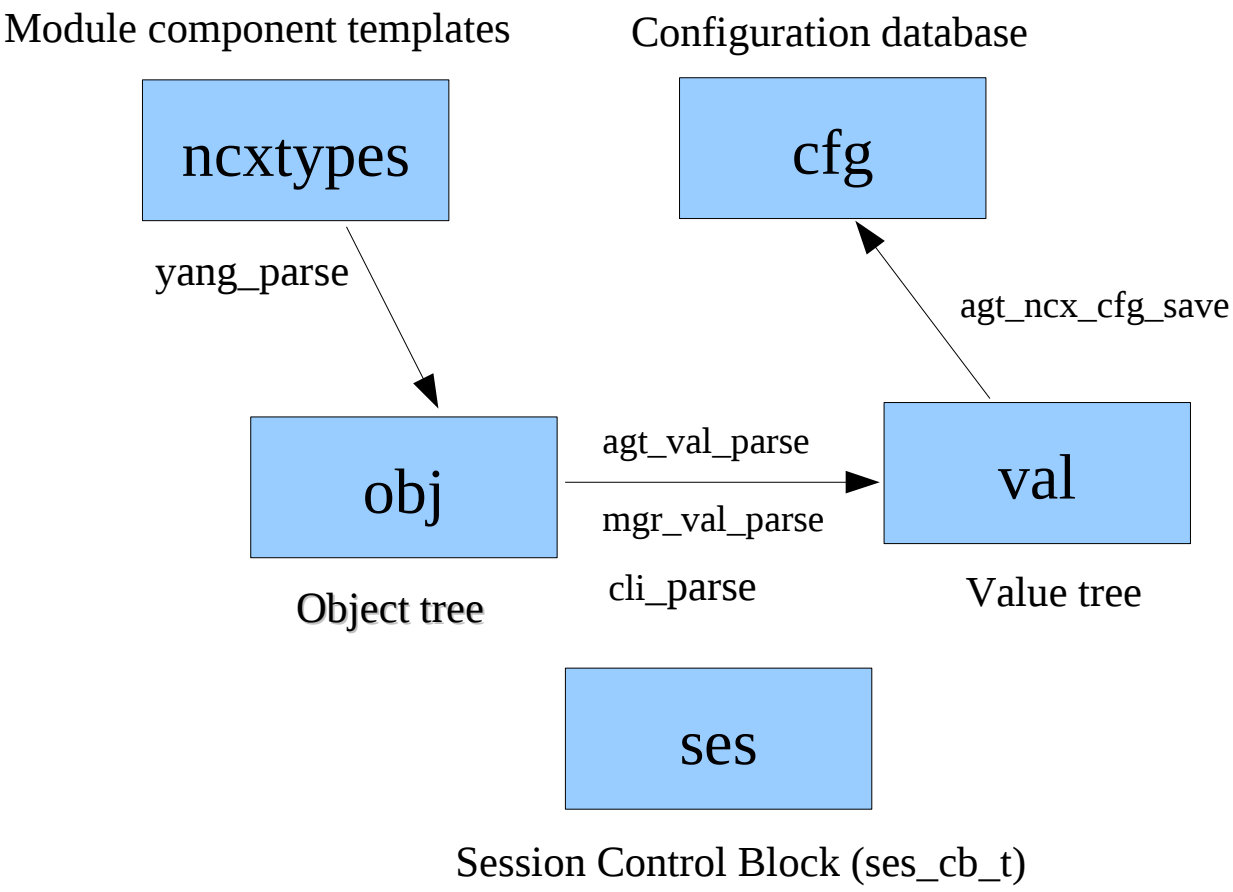
About the Author (2)

- 18 years leading and contributing to Network Management standards efforts in the IETF
  - » RMONMIB WG Chair and co-author (12 years)
    RMON2-MIB, DSMON-MIB, APM-MIB, TPM-MIB
  - » Entity MIB WG Author (5 years)
    ENTITY-MIB (1, 2, 3), ENTITY-SENSOR-MIB
  - » PSAMP (Packet Smapling) WG co-Chair (4 years)
    PSAMP Architecture, Sampling, Information Model
  - » NETCONF WG Co-Chair (5 years)
    NETCONF protocol and transport mappings
  - » NETMOD WG (2 years)
    Major contributor to YANG documents
    Author of YANG Usage Guidelines

Diagnostic Code

- SET_ERROR macro
  - » Used to generate programming exceptions for debugging
  - » All input parameters checked with SET_ERROR for all external functions (if DEBUG flag is set in make)
- Memory leak detection
  - » Internal memory trace code makes sure no memory leaks are left in the code
  - » mtrace functionality built-in (if MEMCHECK flag set in make)
- Centralized error logging facility with printf-style access functions
  - » 6 levels (error, warn, info, debug, debug2, debug3)

YANG Database



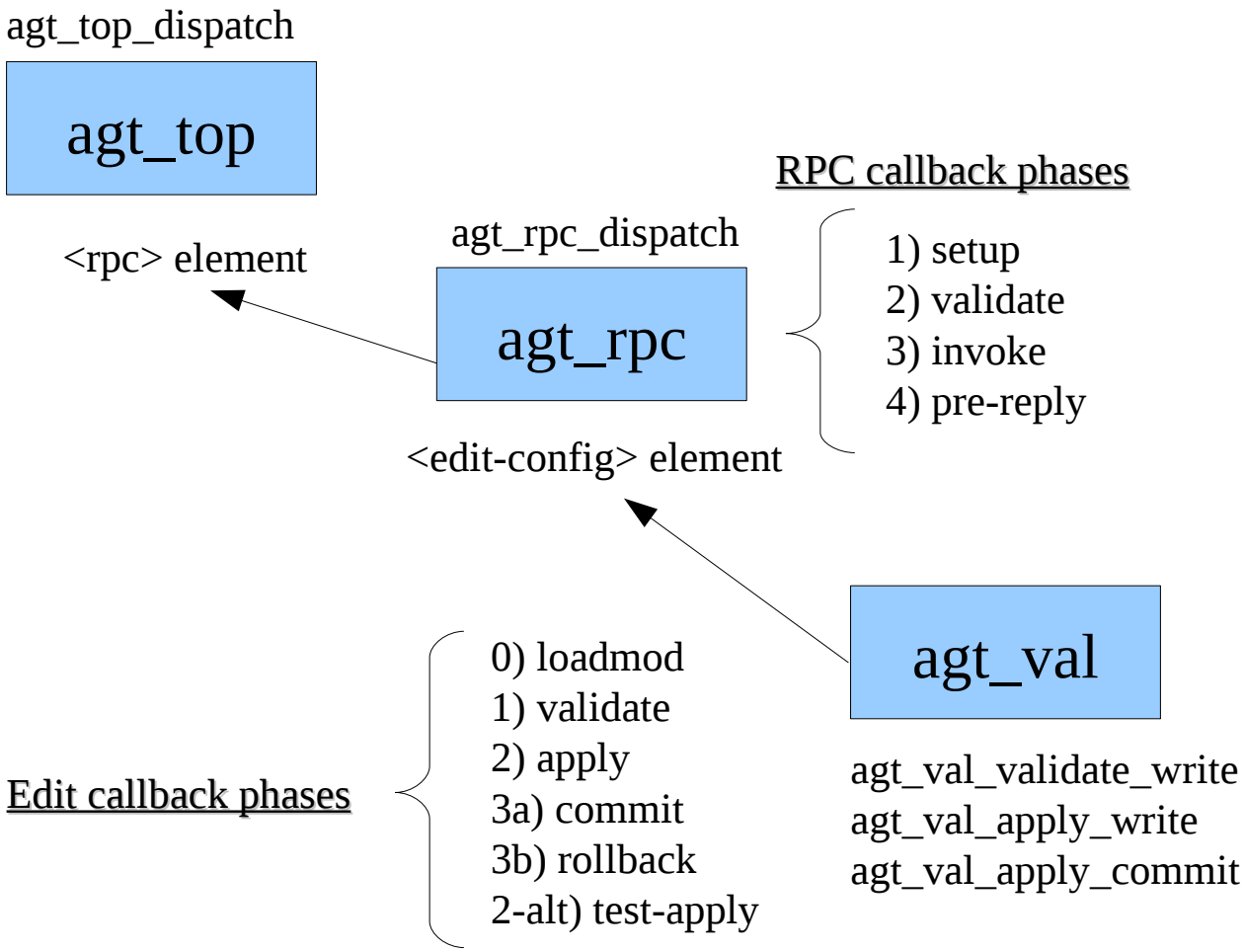
NCX Core Library

- Supports all common data structures
- YANG parsing, convert to internal object tree
- Support for all YANG datatypes and constructs
- YANG module library/revision handler
- XPath 1.0 implementation
- Hash table definition registry
- Double linked queues
- CLI, conf-file, and help text handling
- XML support and XMLNS registry
- Common error handling, logging and diagnostics

netconf/src/ncx files

```
b64.c       dlq.h       obj.c       ses_msg.h   xml_msg.c     yangconst.h
b64.h       ext.c       obj.h       status.c    xml_msg.h     yang_ext.c
blob.c      ext.h       obj_help.c  status.h    xmlns.c       yang_ext.h
blob.h      getcb.h     obj_help.h  tk.c        xmlns.h       yang_grp.c
bobhash.c   grp.c       op.c        tk.h        xml_util.c    yang_grp.h
bobhash.h   grp.h       op.h        top.c       xml_util.h    yang.h
cap.c       help.c      rpc.c       top.h       xml_val.c     yang_obj.c
cap.h       help.h      rpc_err.c   tstamp.c    xml_val.h     yang_obj.h
cfg.c       log.c       rpc_err.h   tstamp.h    xml_wr.c      yang_parse.c
cfg.h       log.h       rpc.h       typ.c       xml_wr.h      yang_parse.h
cli.c       Makefile    runstack.c  typ.h       xpath1.c      yang_typ.c
cli.h       ncx.c       runstack.h  val.c       xpath1.h      yang_typ.h
conf.c      ncxconst.h  send_buff.c val.h       xpath.c
conf.h      ncx.h       send_buff.h val_util.c  xpath.h
def_reg.c   ncxmod.c    ses.c       val_util.h  xpath_yang.c
def_reg.h   ncxmod.h    ses.h       var.c       xpath_yang.h
dlq.c       ncxtypes.h  ses_msg.c   var.h       yang.c
```

Agent Callback Structure



Agent Core Library

- Provides all NETCONF agent functionality
- Key modules:
  - » agt: common init and cleanup
  - » agt_cap/agt_hello: agent capabilities handling
  - » agt_ncx: NETCONF RPC operations
  - » agt_connect/agt_ncxserver/agt_ses: session handling
  - » agt_rpc: <rpc> and <rpc-reply> handler
  - » agt_val/agt_val_parse/agt_xml: content handling
  - » agt_tree/agt_xpath: subtree and Xpath filtering
  - » agt_cb: <edit-config> callback handling
  - » agt_state: ietf-netconf-state module implementation
  - » agt_acm: nacm access control module implementation

netconf/src/agt files

```
agt_acm.c     agt_connect.c     agt_rpc.c      agt_state.h   agt_val.c
agt_acm.h     agt_connect.h     agt_rpcerr.c   agt_timer.c   agt_val.h
agt.c         agt.h             agt_rpcerr.h   agt_timer.h   agt_val_parse.c
agt_cap.c     agt_hello.c       agt_rpc.h      agt_top.c     agt_val_parse.h
agt_cap.h     agt_hello.h       agt_ses.c      agt_top.h     agt_xml.c
agt_cb.c      agt_ncx.c         agt_ses.h      agt_tree.c    agt_xml.h
agt_cb.h      agt_ncx.h         agt_signal.c   agt_tree.h    agt_xpath.c
agt_cli.c     agt_ncxserver.c   agt_signal.h   agt_util.c    agt_xpath.h
agt_cli.h     agt_ncxserver.h   agt_state.c    agt_util.h    Makefile
```

Agent Code Generation

- yangdump --format=h --module=foo
  - » Generates constants for features and identifiers

```
#define ipfix_psamp_F_psampSampRandOutOfN 1
#define ipfix_psamp_N_exportingProcess (const xmlChar *)"exportingProcess"
#define ipfix_psamp_N_template (const xmlChar *)"template"
#define ipfix_psamp_N_observationDomainId (const xmlChar *)"observationDomainId"
```

  - » Generates C equivalent typedefs for all data objects

```
/* list /ipfix/exportingProcess */
typedef struct ipfix_psamp_T_exportingProcess_ {
    dlq_hdr_t qhdr;
    xmlChar *name;
    uint32 exportingProcessId;
    dlq_hdr_t destination;
    dlq_hdr_t fileWriter;
    dlq_hdr_t options;
    dlq_hdr_t template;
    dlq_hdr_t transportSession;
} ipfix_psamp_T_exportingProcess;
```

Manager Core Library

- Provides all NETCONF manager functionality
- Key modules:
  - » mgr: common init and cleanup
  - » mgr_cap/mgr_hello: manger capabilities handling
  - » mgr_io/mgr_ses: session handling
  - » mgr_top: top level element registry
  - » mgr_rpc: <rpc> and <rpc-reply> handler
  - » mgr_val_parse/mgr_xml: content handling

netconf/src/mgr files

```
Makefile   mgr.h        mgr_io.h   mgr_ses.h     mgr_top.h        mgr_xml.h
mgr.c      mgr_hello.c  mgr_rpc.c  mgr_signal.c  mgr_val_parse.c
mgr_cap.c  mgr_hello.h  mgr_rpc.h  mgr_signal.h  mgr_val_parse.h
mgr_cap.h  mgr_io.c     mgr_ses.c  mgr_top.c     mgr_xml.c
```

Summary: Part 3

- The NCX NETCONF/YANG implementation is complete and ready to be adapted to many platforms
- Security, protocol conformance, and automation are built-in features from the ground up
- Many content-based add-on features are possible, which would continue to increase toolkit value over time