YumaPro SDK Developer Training

Andy Bierman
andy@yumaworks.com
2015-10-15
Agenda

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Part 1: Intro/Big Picture

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Introduction

- Remote Control
- Configuration, Operational Data and Notifications
- Evolution of CLI, SNMP into NETCONF
- What are the standards to know?
- What are the YumaPro Tools?
- Install YumaPro SDK
Remote Control

• Asynchronous network management
  – Management application uses a shared or dedicated network to communicate with managed devices
  – Security set up in advance
    • Usually console login or flash drive
  – Management protocols send commands for specific data
    • Client and server must agree on the syntax and semantics of this data
Configuration

- Called the intended configuration
  - Instructions to the device
    - Some vendors this is only client instructions
    - YANG: config=true data nodes
- Transferable
  - Load config into new device to clone
- Offline validation and manipulation
  - Get config, edit offline, push config
- EntityTag and LastModified meta-data maintained by the server
Operational State

- YANG: config=false data nodes
- Divided into
  - Applied configuration
  - Statistics
  - Derived state
- No EntityTag or LastModified meta-data maintained by the server
Notifications

- YANG: notification statement
- Data model specific events
- Not a subscription to a resource that has changed
  - Pub/sub updates resources
  - YANG notifications report events
- Not private
  - Any subscriber can access all events; no private events for 1 session
Evolution

- CLI over telnet was first; SSH came later
- CLI mandatory to provide complete functional coverage
- SNMP added monitoring based on data models
- Configuration in CLI and monitoring in SNMP causes problems (data naming, RowStatus, etc.)
- YANG is a better fit for configuration
- Large effort in progress to standardize YANG-based configuration and monitoring data models
Standards

- Current
  - NETCONF/YANG
  - XML/XPath
- Almost Here
  - RESTCONF/JSON
- Near Future
  - CoMI/CBOR/YANG Hash
YumaPro SDK
Core Tools

- SMIv2 MIB module
- Smidump translator
- YANG module
- Yangdump-sdk YANG translator
- Text test scripts
- Yangcli-pro client
- Netconfd-pro server
- C code for server development
- Network
Download Packages

- https://www.yumaworks.com/dld
  - user: training
  - password: see whiteboard
  - yumapro-sdk/14.08-9
  - yumapro-doc/14.08-9
Installing YumaPro

- Refer to YumaPro SDK Installation Guide
- Install library dependencies
- Install yumapro-sdk and yumapro-doc packages
- Update SSH and Apache configurations
- Install netconfd-pro.conf

YumaPro Installation Guide
netconfd-pro.conf

- Server configuration file
  - default: /etc/yumapro/netconfd-pro.conf
  - CLI parameters override config file values
  - --config=filespec overrides default config file

- Start config file from sample
  - cd /etc/yumapro
  - sudo cp netconfd-pro-sample.conf
    netconfd-pro.conf
Running YumaPro

- Start the server
  - netconfd-pro –log-level=debug4
- Start yangcli-pro and connect to the server
- Try some commands like get, get-config
- Special files
  - $HOME/.yumapro files
  - /tmp/ncxserver.sock
  - YUMAPRO_HOME/modules
  - /usr/share/yumapro
  - /usr/lib/yumapro
Getting Help

• Several sources of help (besides the manuals!)
  – netconfd-pro
    • netconfd-pro –help
    • man netconfd-pro
  – yangcli-pro
    • yangcli-pro –help
    • man yangcli-pro
    • help command
    • tab key for command completions
    • ? key for command completion help
Documentation

- /usr/share/doc/yumapro
  - YumaPro Installation Guide
  - YumaPro Quickstart Guide
  - YumaPro User Manual
  - YumaPro netconfd-pro Manual
  - YumaPro yangdiff-pro Manual
  - YumaPro yangdump-pro Manual
  - YumaPro Developer Manual
  - YumaPro yp-system API Guide
  - YumaPro yp-show API Guide
Part 2: YANG Tutorial

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
• Switch to Netconf Central slide deck
  • http://dld.netconfcentral.org/doc/ncorg_netconf_yang_tutorial.pdf
YumaPro YANG Processing

YANG --> Internal Data

- YANG modules
- deviations
- annotations
- configuration

libncx

Data structures
Part 3: Getting Started

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Server Development

- Pick 1 or more YANG modules to implement
- Decide which type of SIL variant is needed
- Generate the C Code Stubs
- Fill in the callback functions
Server Instrumentation Library (SIL) code is protocol-independent so engineers can focus on managing the system, not all the management protocols.
YumaPro SDK
server: netconfd-pro
YumaPro Server
Major Features

- Redesigned for Multi-protocol support
  - CLI, NETCONF, REST APIs
  - XML and JSON encoding
- Distributed Transaction Engine
  - Full YANG validation, optimized for speed
  - Local (SIL) and remote (SIL-SA) callbacks
  - Automatic complex rollback
- Distributed System Integration
  - System Integration Library
  - Multiple Sub Agents
  - External Database
YumaPro Server
Standards Based

- Full NETCONF standards support
  - NETCONF Protocol (RFC 6241)
  - NETCONF over SSH (RFC 6242)
  - NETCONF Access Control Model (RFC 6536)
  - NETCONF Monitoring (RFC 6022)
  - NETCONF Notifications (RFC 5277)
  - NETCONF Base Notifications (RFC 6470)
  - Partial Lock RPC for NETCONF (RFC 5717)
  - With-defaults Capability (RFC 6243)
  - YANG Data Modeling Language (RFC 6020)
  - Common YANG Data Types (RFC 6991)
Regular YANG modules are used to create code stubs:
- system
- user
Developers only focus on development of product specific functions:
- database manipulation
- device instrumentation
- undo device instrumentation
YumaWorks Solves Management Plane Automation

The user and system code is then built into a dynamically loadable library – a SIL (Server Instrumentation Library) and for distributed Systems a SIL-SA (SIL-Sub Agent).
Bundles can group collections of YANG modules together into a dynamic library that can be loaded with one command.
SIL Code

• Server Instrumentation Library
  – Implements user callbacks for YANG objects
  – Used to hook data model into the underlying system
  – When server loads a module or bundle, it looks for the associated SIL library (using naming conventions)
  – libdy (dynamic load) used to avoid compile-time coupling to SILs
Distributed Transaction Management

Asynchronous Sub-Agent Support (SIL-SA)
SIL vs. SIL-SA Code

- **SIL== Main Server**
  - Synchronous instrumentation

- **SIL-SA – Sub-Agent**
  - Asynchronous (distributed) instrumentation
  - Same as SIL, except runs in a separate process over the YControl protocol
Module vs. Bundle

- A SIL bundle contains SIL or SIL-SA code for multiple YANG modules
  - Needed in order to auto-generate C code for augments
  - Allows multiple modules to be loaded into the server as a single package
    - --bundle=ietfInterfacesBundle
**make_sil_dir Scripts**

- Scripts to generate the instrumentation library:
  - `make_sil_dir_pro`
    - 1 module, SIL or SIL-SA code
  - `make_sil_bundle`
    - N modules, SIL or SIL-SA code
make_sil_dir_pro

- 1 module for SIL or SIL-SA code
  - make_sil_dir_pro [--split] module-name [extra-params]
  - extra-params
    - deviation=deviation-module-name
    - --sil-sa
    - --sil-get2
    - --sil-edit2
    - --sil-include=include-spec
    - other yangdump-sdk parameters

Developer Sec. 2, SDK Quickstart
SIL Example

address.yang → address

- Makefile
  - bin
  - src
  - lib

Makefile
  - y_address.c
  - y_address.h
  - u_address.c
  - u_address.h
make_sil_bundle

- N modules for SIL or SIL-SA code
  - make_sil_bundle bundle-name module-name [module-name]* [extra-parms]
  - base module first, augmenting modules last
  - extra-parms
    - deviation=deviation-module-name
    - --sil-sa
    - --sil-get2
    - --sil-edit2
    - --sil-include=include-spec
    - other yangdump-sdk parameters

Developer Sec. 2, SDK Quickstart
SIL Bundle Example

make_sil_bundle test1 t60 t61 t62

t60.yang  t61.yang  t62.yang

test1

Makefile

bin  src  lib

Makefile

y_test1.c  y_t61.c  y_t62.c
y_test1.h  y_t61.h  y_t62.h
y_t60.c  y_t60.h  u_t61.c  u_t61.h  u_t62.c  u_t62.h
u_t60.c  u_t60.h
**make_sil_sa_dir**

- 1 module for SIL-SA code
  - `make_sil_sa_dir [--split] module-name [extra-parms]`
  - extra-parms
    - deviation=`deviation-module-name`
    - --sil-get2
    - --sil-edit2
    - --sil-include=`include-spec`
    - other yangdump-sdk parameters
make_sil_sa_bundle

• N modules for SIL-SA code
  - make_sil_sa_bundle bundle-name
    module-name [module-name]*
    [extra-parms]
  - base module first, augmenting modules last
  - extra-parms
    • deviation=deviation-module-name
    • --sil-get2
    • --sil-edit2
    • --sil-include=include-spec
    • other yangdump-sdk parameters
--split Parameter

• Use --split for separate user and yumapro files

A split SIL module for foo.yang would be generated using the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u_foo.c</td>
<td>User SIL</td>
<td>User-provided server instrumentation code for the 'foo' module.</td>
</tr>
<tr>
<td>u_foo.h</td>
<td>User SIL</td>
<td>User-provided external definitions for the 'foo' module. Should not edit!</td>
</tr>
<tr>
<td>y_foo.c</td>
<td>YumaPro SIL</td>
<td>YumaPro server glue code for the 'foo' module. Do not edit!</td>
</tr>
<tr>
<td>y_foo.h</td>
<td>YumaPro SIL</td>
<td>YumaPro server external definitions for the 'foo' module. Do not edit!</td>
</tr>
</tbody>
</table>
Without --split

- Old style: combined files for full parameter access

A combined SIL module for foo.yang would be generated using the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo.c</td>
<td>Combined YumaPro and User SIL</td>
<td>User-provided server instrumentation code for the foo module.</td>
</tr>
<tr>
<td>foo.h</td>
<td>Combined YumaPro and User SIL</td>
<td>User-provided external definitions for the foo module. Should not edit!</td>
</tr>
</tbody>
</table>
SIL Callbacks

NETCONF Callback Points

- top
- top_register_node
- agt_rpc_register_method
- remote procedure call
- agt_cb_register_callback
- database edit operations
- <my-data>
- <edit-config>
- <copy-config>
- <commit>
- <validate>
**make_sil_dir_pro Example**

- Create code for ietf-interfaces SIL
  - `make_sil_dir_pro --split ietf-interfaces \ --sil-get2 --sil-edit2`
  - `cd ietf-interfaces`
  - `make DEBUG=1`
  - `sudo make DEBUG=1 install`
SIL APIs

- There are 3 functions a SIL or SIL-SA library are required to implement:
  - init1
  - init2
  - cleanup
- init1 is used to load modules, register callbacks, etc.
- init2 is used to load run-time data into the datastore
- cleanup is used to unload the module, free memory, etc.
• YumaPro init1 function registers callbacks

```c
/**
 * FUNCTION y_ietf_interfaces_init
 *
 * initialize the ietf-interfaces server instrumentation library
 *
 * INPUTS:
 * modname == requested module name
 * revision == requested version (NULL for any)
 *
 * RETURNS:
 * error status
 */

status_t y_ietf_interfaces_init (const xmlChar *modname,
                                 const xmlChar *revision)
{
```
User init1 function caches module pointer

```c
status_t u_iarf Interfaces_init (  
    const xmlChar *modname,  
    const xmlChar *revision)  
{
    status_t res = NO_ERR;  
    ncf_module_t *ietf_interfaces_mod = NULL;  

    ietf_interfaces_mod = ncf_find_module(  
        y_iarf_interfaces_M_iarf_interfaces,  
        y_iarf_interfaces_R_iarf_interfaces);  
    if (ietf_interfaces_mod == NULL) {  
        return ERR_NCX_OPERATION_FAILED;  
    }

    /* put your module initialization code here */

    return res;
} /* u_iarf_interfaces_init */
```
- YumaPro init2 sets up system-created config and operational state

```c
static status_t
  agt_if_init2 (void)
{
  if (!agt_if_supported) {
    return NO_ERR;  // ignore module load error
  }

  boolean added = FALSE;
  status_t res = NO_ERR;
  val_value_t *interfacesval =
    agt_add_top_node_if_missing(ifmod, interfaces_N_interfaces,
      &added, &res);
  // ignoring added flag because counter hooks need to be added
  if (interfacesval && res == NO_ERR) {
    res = add_interface_entries(interfacesval);
  }
  return res;
}
/* agt_if_init2 */
• User init2 does implementation-specific tasks

```c
/**************************************************************************/
* FUNCTION u_ietf_interfaces_init2
*
* SIL init phase 2: non-config data structures
* Called after running config is loaded
*
* RETURNS:
* error status
**************************************************************************/

status_t u_ietf_interfaces_init2 (void)
{
    status_t res = NO_ERR;

    /* put your init2 code here */

    return res;

} /* u_ietf_interfaces_init2 */
```
y_cleanup

- YumaPro cleanup unregisters callbacks

```c
void y_ietf_interfaces_cleanup (void)
{
    agt_cb_unregister_callbacks(
        y_ietf_interfaces_M_ietf_interfaces,
        (const xmlChar *)"/if:interfaces";
    
    agt_cb_unregister_callbacks(
        y_ietf_interfaces_M_ietf_interfaces,
        (const xmlChar *)"/if:interfaces/if:interface";
    
    agt_cb_unregister_callbacks(
        y_ietf_interfaces_M_ietf_interfaces,
        (const xmlChar *)"/if:interfaces-state";
}```
• User cleanup does implementation-specific cleanup tasks

/**
 * FUNCTION u_iwf_interfaces_cleanup
 *  cleanup the server instrumentation library
 */

void u_iwf_interfaces_cleanup (void)
{

    /* put your cleanup code here */
}

/* u_iwf_interfaces_cleanup */
Part 4: RPC Processing

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
RPC Processing

- YANG “rpc” statement used to define all operations
- NETCONF <rpc> and <rpc-reply> messages
- SIL init1 function will register each RPC with “agt_rpc_register_method” function
- Multi-phase RPC callback model
- Processing input
- 3 return variants (ok, data, rpc-error)
- Returning data (data queue or walker callback)
- Post Reply Processing
YANG rpc-stmt

```text
rpc-stmt = rpc-keyword sep identifier-arg-str optsep

";" / "{" stmtsep

;; these stmts can appear in any order
*(if-feature-stmt stmtsep)
[status-stmt stmtsep]
[description-stmt stmtsep]
[reference-stmt stmtsep]
*(typedef-stmt /
   grouping-stmt) stmtsep)
[input-stmt stmtsep]
[output-stmt stmtsep]
"}"

input-stmt = input-keyword optsep

{" stmtsep

;; these stmts can appear in any order
*(typedef-stmt /
   grouping-stmt) stmtsep)
1*(data-def-stmt stmtsep)
"}"

output-stmt = output-keyword optsep

{" stmtsep

;; these stmts can appear in any order
*(typedef-stmt /
   grouping-stmt) stmtsep)
1*(data-def-stmt stmtsep)
"}"
```
<rpc> Message

- Sent by client to server to request an operation

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users>
          <user>
            <name>fred</name>
          </user>
        </users>
      </top>
    </filter>
  </get-config>
</rpc>
```
<rpc-reply> Message

- Sent by the server to the client and contains the status and/or data for the specific RPC request

```xml
<rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top xmlns="http://example.com/schema/1.2/config">
      <users>
        <user>
          <name>fred</name>
          <type>admin</type>
          <full-name>Fred Flintstone</full-name>
          <company-info>
            <dept>2</dept>
            <id>2</id>
          </company-info>
        </user>
        <!-- More users -->
      </users>
    </top>
  </data>
</rpc-reply>
```
agt_rpc Module

- **init_fn:**
  - agt_rpc_register_method
  - agt_rpc_support_method
  - agt_rpc_unsupport_method

- **cleanup function:**
  - agt_rpc_unregister_method
RPC Callbacks

5 Phase RPC Processing Model

agt_ses → agt_top

<rpc>

agt_rpc

PARSE → VALIDATE → INVOKE → REPLY → POST

Optional Callbacks

validate callback

invoke callback

post-reply callback
RPC Callback Function

- All phases have same function footprint

```c
/ ***************************************************************
 * FUNCTION y_iETF_netconf_partial_lock_partial_lock_validate 
 *
 * RPC validation phase
 * All YANG constraints have passed at this point.
 * Add description-stmt checks in this function.
 *
 * INPUTS:
 * see agt/agt_rpc.h for details
 *
 * RETURNS:
 * error status
 ***************************************************************/
static status_t
  y_iETF_netconf_partial_lock_partial_lock_validate ( 
    ses_cb_t *scb, 
    rpc_msg_t *msg, 
    xml_node_t *methnode)
{
```

Sec. 6.4, RPC Operation Interface
RPC Validate Phase

PARSE → VALIDATE → INVOKE → REPLY → POST REPLY

validate callback → invoke callback

static status_t
y_toaster_make_toast_validate (ses_cb_t *scb,
rpc_msg_t *msg,
xml_node_t *methnode);

return operation status

session making the request

RPC request in progress

XML node for the operation,
for error reporting purposes
and vendor-specific attributes
RPC Invoke

RPC Invoke Phase

- PARSE
  - validate callback
- VALIDATE
  - static status_t return operation status
  - y_toaster_make_toast_invoke (ses_cb_t *scb, rpc_msg_t *msg, xml_node_t *methnode);
- INVOKE
  - invoke callback
- REPLY
  - post-reply callback
  - session making the request
  - validated RPC request
  - XML node for the operation, for error reporting purposes and vendor-specific attributes
RPC Post Reply Phase

PARSE → VALIDATE → INVOKE → REPLY → POST REPLY

validate callback → invoke callback → post-reply callback

static status_t
y_foo_command_post (ses_cb_t *scb,
rpc_msg_t *msg,
xml_node_t *methnode);

return status should be NO_ERR
session making the request
completed RPC request
XML node for the operation, ignored in this mode

Copyright © 2015, YumaWorks, Inc. All Rights Reserved. Company Confidential
Part 5: Notifications

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Notifications

- YANG “notification” statement defines event name, event namespace, and payload
- NETCONF <notification> message
- SIL “send” function generated for each notification
- Notification “send” functions do not register any callback, since they initiate the event notification
- <create-subscription> operation
- Notification CLI Parameters
- YumaPro notification extensions
YANG notification-stmt

- Notification data-def-stmt used in callback

```py
notification-stmt = notification-keyword sep
                   identifier-arg-str optsep
                   (";": /
                    "{" stmtsep
                    ;; these stmts can appear in any order
                    *(if-feature-stmt stmtsep)
                    [status-stmt stmtsep]
                    [description-stmt stmtsep]
                    [reference-stmt stmtsep]
                    *((typedef-stmt /
                                   grouping-stmt) stmtsep)
                    *(data-def-stmt stmtsep)
                    ")"
```
YANG notification-stmt
Example

notification netconf-capability-change {

description "Generated when the NETCONF server detects that
the server capabilities have changed.
Indicates which capabilities have been added, deleted,
and/or modified. The manner in which a server
capability is changed is outside the scope of this
document.";

uses changed-by-parms;
}

leaf-list added-capability {

type inet:uri;

description "List of capabilities that have just been added.";
}

leaf-list deleted-capability {

type inet:uri;

description "List of capabilities that have just been deleted.";
}

leaf-list modified-capability {

type inet:uri;

description "List of capabilities that have just been modified.
A capability is considered to be modified if the
base URI for the capability has not changed, but
one or more of the parameters encoded at the end of
the capability URI have changed.
The new modified value of the complete URI is returned.";
}

} // notification netconf-capability-change
Notification Structure

The `<notification>` element is the top-level NETCONF message

```markdown
notification {
  eventTime 2010-01-01T19:17:33Z
  sysConfigChange {
    userName andy
    sessionId 1
    remoteHost 127.0.0.1
    edit {
      target /toast:toaster
      operation create
    }
  }
  sequence-id 4
}
```

The `<sequence-id>` element is added to every `netconfd-pro` notification if profile->agt_notif_sequence_id set

The notification event payload is defined by a YANG notification statement
SIL notif_send Function

- Parameters based on the YANG payload
  - Same function for y_ and u_ variants

```c
void u_ietf_netconf_notifications_netconf_capability_change_send (
  y_ietf_netconf_notifications_T_netconf_capability_change_changed_by *changed_by,
  const xmlChar *added_capability,
  const xmlChar *deleted_capability,
  const xmlChar *modified_capability)
{
  // Implementation...
}
```

Developer Sec. 6.6, Notifications
SIL notif_send Tasks

- Check if notifications enabled or exit
- Create a new notification
  - agt_not_new_notification
- Add data to payload
  - agt_not_add_to_payload
- Queue the notification for processing
  - SIL: agt_not_queue_notification
  - SIL-SA: sil_sa_queue_notification
RFC 5277 defines NETCONF notifications
  - 1 subscription per session
  - :interleave capability if commands allowed on the subscription session
  - netconfd-pro never times out a session receiving notifications
  - netconfd-pro always advertises :interleave

- filter parameter (subtree or XPath)
- Replay Support
  - startTime and stopTime parameters allowed
Replay Buffer

Notification Replay Buffer

System Event Log

default eventlog-size = 1000

1 2 3 4 5 6 ...

Replay 1 (stops when session is closed)

Replay 2 (finite)

Replay 3 (stops in the future)

Replay 1: startTime=2009-01-01, no stopTime
Replay 2: startTime=2009-12-01, stopTime=2009-12-31
Replay 3: startTime=2009-10-01, stopTime=2011-12-31
Notification CLI Parameters

- `--eventlog-size=number [d: 1000]`
  - replay buffer size
- `--max-burst=number [d: 10]`
  - max notifications to 1 session per second
- `--message-indent=number [d: -1]`
  - affects all server sent messages
- `--with-notifications=boolean [d: true]`
  - used to disable notifications
YumaPro Notification Extensions

• sequence-id:
  – Enabled in server profile at boot-time
  – Adds incremental numeric ID to each notification event
  – Sequence-id is per event generated, not per message sent to all subscription sessions

• yumaworks-event-filter.yang
  – Suppress generation of specific event types
  – <filter> is post-replay buffer
  – <event-filter> is pre-replay buffer

• <cancel-subscription> operation
Part 6: Retrieval Operations

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Retrieval Operations

- YANG data definition statement used to specify syntax and semantics of all conceptual data structures on the server
- Configuration Nodes (val_value_t trees)
- 3 ways to implement retrieval callbacks
- Defaults handling
- Entity tags and Last-Modified timestamps
- YumaPro retrieval extensions
YANG data-def-stmt

data-def-stmt = container-stmt /
                leaf-stmt /
                leaf-list-stmt /
                list-stmt /
                choice-stmt /
                anyxml-stmt /
                uses-stmt
config-stmt

- config=true (default)
  - configuration datastore contents
  - <get-config>, <edit-config>, etc.
- config=false
  - operational data
  - <get>
3 GET Callback Variants

- 3 different ways to support operational data
  - static get
    - data and value exist in data tree
  - virtual get
    - data exists in data tree but value retrieved via callback
      - callback stored in each virtual node
  - get2
    - no data exists in the data tree
      - callback stored in object schema node
Static GET

- **static get**
  - Data node exists in the system data tree
  - Value stored in the data node
  - All config=true nodes are currently this type
  - This variant is suitable for operational data that is static, such as the system boot-time
Static GET Registration

- Example GET static implementation
  - Generate the static value
  - Create a static leaf with this value
  - add child to parent

```c
/* add /system/sysBootDateTime */
xmlChar tstampbuff[TSTAMP_MIN_SIZE];
tstamp_datetime(tstampbuff);
childval = agt_make_leaf(systemobj,
    system_N_sysBootDateTime,
    tstampbuff, &res);
if (childval) {
    val_add_child(childval, topval);
} else {
    return res;
}
```
Virtual GET

- virtual get1
  - Data node exists in the system data tree
  - Value NOT stored in the data node
  - Uses get callback
  - Created with utility functions:
    - val_init_virtual
    - agt_make_virtual_leaf
  - Data retrieval done by server as needed
    - val_get_virtual_value
Virtual GET Registration

• Example Virtual Get Registration
  – create a leaf with a callback
  – add to the parent

```c
/* add /system/sysCurrentDateTime */
childval = agt_make_virtual_leaf(systemobj,
  system_N_sysCurrentDateTime,
  get_currentDateTime, &res);

if (childval) {
  val_add_child(childval, topval);
} else {
  return res;
}
```
Virtual GET Callback

• Example Virtual Get Callback

```c
static status_t
get_currentDateTime (ses_cb_t *scb,
getcb_mode_t cbmode,
const val_value_t *virval,
val_value_t *dstval)
{
    (void)scb;
    (void)virval;

    if (cbmode == GETCB_GET_VALUE) {
        xmlChar *buff = (xmlChar *)m__getMem(TSTAMP_MIN_SIZE);
        if (!buff) {
            return ERR_INTERNAL_MEM;
        }
        tstamp_datetime(buff);
        VAL_STR(dstval) = buff;
        return NO_ERR;
    } else {
        return ERR_NCX_OPERATION_NOT_SUPPORTED;
    }

} /* get_currentDateTime */
```
GET2 Variant for Retrieval

- get2
  - Data nodes are not in the system data tree
  - get2 callback registered for container, list, and choice schema nodes
  - Callback returns child terminal nodes, not entire subtree
  - Keys, select nodes, and content match nodes available to callback in getcb_get2_t struct
  - Get, GetNext, and GetBulk modes supported
  - Callback usually returns NO_ERR if data found of ERR_NCX_NO_INSTANCE if the requested data is not found
Callback Tasks for GET2

- non-presence (NP) container
  - return requested child nodes
- presence (P) container
  - check if container exists
  - returned requested child nodes
- choice
  - return active-case name (if any)
  - return requested child nodes
- list
  - get requested instance(s), add return keys
  - return requested child nodes
GET2 Registration

- 1 Callback registered per complex object

```c
res = agt_cb_register_get_callback(
    y_ietf_interfaces_M_ietf_interfaces,
    (const xmlChar *)"/if:interfaces-state",
    y_ietf_interfaces_R_ietf_interfaces,
    ietf_interfaces_interfaces_state_get);
if (res != NO_ERR) {
    return res;
}
```

```c
res = agt_cb_register_get_callback(
    y_ietf_interfaces_M_ietf_interfaces,
    (const xmlChar *)"/if:interfaces-state/if:interface",
    y_ietf_interfaces_R_ietf_interfaces,
    ietf_interfaces_interfaces_state_interface_get);
if (res != NO_ERR) {
    return res;
}
```
**y_ GET2 Callback**

- YumaPro GET2 Callback is always the same

```c
/****************************************************************************
* FUNCTION ietf_interfaces_interfaces_state_interface_statistics_get
* *
* Get database object callback for container statistics
* Path: /interfaces-state/interface/statistics
* Fill in 'get2cb' response fields
* *
* INPUTS:
*   see ncx/getcb.h for details (getcb_fn2_t)
* *
* RETURNS:
*   error status
****************************************************************************/
static status_t ietf_interfaces_interfaces_state_interface_statistics_get (  
    ses_cb_t *scb,
    xml_msg_hdr_t *msg,
    getcb_get2_t *get2cb)
{
```
u_ GET2 Callback

- User GET2 callback parameters include the get2cb and any ancestor keys

```c
/**
 * FUNCTION u_ietchInterfaces_interfaces_state_interface_statistics_get
 * 
 * Get database object callback for container statistics
 * Path: /interfaces-state/interface/statistics
 * Fill in 'get2cb' response fields
 * 
 * INPUTS:
 *  see ncx/getcb.h for details (getcb_fn2_t)
 * 
 * RETURNS:
 * error status
 */

status_t u_ietchInterfaces_interfaces_state_interface_statistics_get (getcb_get2_t *get2cb,
                       const xmlChar *k_interfaces_state_interface_name)
{
    ...
    ...
```
GET2 Callback
GET or GETNEXT Mode

• For container or choice, only GET allowed
  /* check the callback mode type */
  getcb_mode_t cbmode = GETCB_GET2_CBMODE(get2cb);
  switch (cbmode) {
    case GETCB_GET_VALUE:
      break;
    case GETCB_GETNEXT_VALUE:
      return ERR_NCX_NO_INSTANCE;
    default:
      return SET_ERROR(ERR_INTERNAL_VAL);
  }

• For list, GET or GETNEXT is allowed
  boolean getnext = FALSE;

  /* check the callback mode type */
  getcb_mode_t cbmode = GETCB_GET2_CBMODE(get2cb);
  switch (cbmode) {
    case GETCB_GET_VALUE:
      break;
    case GETCB_GETNEXT_VALUE:
      getnext = TRUE;
      break;
    default:
      return SET_ERROR(ERR_INTERNAL_VAL);
  }
GET2 Callback Match Test

- Optional subtree filter content-match test for non-key child terminal nodes
  - only return instance (e.g. row) if the children matching the specified content-match nodes exist with the same values

```c
/* optional: check if any content-match nodes are present */
boolean match_test_done = FALSE;
val_value_t *match_val = GETCB_GET2_FIRST_MATCH(get2cb);
for (; match_val; match_val =
    GETCB_GET2_NEXT_MATCH(get2cb, match_val)) {

    //**** CHECK CONTENT NODES AGAINST THIS ENTRY ***/

} GETCB_GET2_MATCH_TEST_DONE(get2cb) = match_test_done;
```
GET2 Callback List Keys

- List keys passed from y_get to u_get in triplet
  - value, fixed-flag, present-flag

```c
/* *******************************************************************************/
* FUNCTION u_ietch_interfaces_interfaces_state_interface_get
* *
* Get database object callback for list interface
* Path: /interfaces-state/interface
* Fill in 'get2cb' response fields
*
* INPUTS:
*   see ncx/getcb.h for details (getcb_fn2_t)
* *
* RETURNS:
*   error status
*******************************************************************************/

status_t u_ietch_interfaces_interfaces_state_interface_get (  
  getcb_get2_t *get2cb,
  const xmlChar *k_interfaces_state_interface_name,
  boolean name_fixed,
  boolean name_present)
{
  
```
GET2 Callback
List Key fixed-flag

- If set for a key then do not increment for GETNEXT
  - E.g., 'name' key would be set for this request

```xml
<filter type="subtree">
  <top xmlns="http://example.com/schema/1.2/config">
    <users>
      <user>
        <name>fred</name>
      </user>
    </users>
  </top>
</filter>
```
GET2 Callback

List Key present-flag

- If false for a key then return the first entry
- The first request for a list may be a GET with no keys
  - Return the first instance
- E.g., a request for an entire list:

```xml
<filter type="subtree">
  <top xmlns="http://example.com/schema/1.2/config">
    <users/>
  </top>
</filter>
```
GET2 Callback

List Key return keys

- The key leafs for each entry are returned by the GET2 callback

  - No auto-generated code to handle finding next instance to return

```c
/* for GET, make sure all local keys present */
if (!getnext && !name_present) {
    return ERR_NCX_NO_INSTANCE;
}

/* For GET, find the entry that matches the key values
 * For GETNEXT, find the entry that matches the next key value
 * If the 'present' flag is false then return first key instance
 * If the 'fixed' flag is true then no GETNEXT advance for the key
 * Create a new return key val_value_t, then getcb_add_return_key */

/***** ADD RETURN KEYS AND REMOVE THIS COMMENT *****/

if (GETCB_GET2_FIRST_RETURN_KEY(get2cb) == NULL) {
    return ERR_NCX_NO_INSTANCE;
}
```
The more_data flag is set by list callbacks
  - Better to return 'true' if not sure

```c
/* For GETNEXT, set the more_data flag true if not sure */
boolean more_data = (getnext) ? TRUE : FALSE;

/*** SET more_data FLAG ****/
GETCB_GET2_MORE_DATA(get2cb) = more_data;
```
GET2 Callback
Requested Nodes

- Only applies if there are child terminal nodes

```c
/* go through all the requested terminal child objects */
obj_template_t *childobj =
getcb_first_requested_child(get2cb, obj);
for (; childobj; childobj =
getcb_next_requested_child(get2cb, childobj)) {

    /* Retrieve the value of this terminal node and
       * add with getcb_add_return_val */

    /* leaf type (identityref) */

    /* leaf admin-status (enumeration) */

    /* leaf oper-status (enumeration) */

    /* leaf last-change (string) */
```
GET2 SIL Code Exercise

- Examine u_ietf-interfaces.c
- Add code to find interface instances
- Add code to retrieve interface counters
- Build and load SIL library
- Test counter retrieval
Part 7: Transaction Model

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Transaction Model

- Datastores
- Multiple Transaction Variants
- Datastore Locking
- Confirmed Commit Procedure
What is a Datastore?

- A Datastore is not a Database
- A conceptual collection of top-level YANG data nodes
  - All datastores share the same schema set
    - Not all protocols access all datastores
  - All data instances in the datastore share some common properties
- Current NETCONF datastores contain only config=true data nodes
3 Standard Datastores

- **running** (mandatory)
  - represents current intended configuration
- **candidate** (optional :candidate capability)
  - represents pending edits, not yet applied to running
    - `<discard-changes>` to clear candidate of edits
    - `<commit>` to apply all the edits to running
      - mandatory all-or-none commit
- **startup** (optional :startup capability)
  - represents read-only intended configuration for the next reboot
Transaction Variants

:writable-running

edit-config  →  <running>  →  Automatic NV-save

copy-config

:writable-running + :startup

edit-config  →  <running>  →  <startup>

copy-config

:candidate

edit-config  →  <candidate>  →  <running>  →  Automatic NV-save

copy-config

:candidate + :startup

edit-config  →  <candidate>  →  <running>  →  <startup>

copy-config
Transaction Variants

CLI Options

:writable-running

  --target=running
  --with-startup=false

:writable-running + :startup

  --target=running
  --with-startup=true

:candidate

  --target=candidate
  --with-startup=false

:candidate + :startup

  --target=candidate
  --with-startup=true
Datastore Locking

• Global <lock> and <unlock>
  – 1 datastore at a time
  – need to lock all datastores
  – Automatic lock release if session for lock-owner terminates for any reason

A lock MUST NOT be granted if any of the following conditions is true:

* A lock is already held by any NETCONF session or another entity.
* The target configuration is <candidate>, it has already been modified, and these changes have not been committed or rolled back.
* The target configuration is <running>, and another NETCONF session has an ongoing confirmed commit (Section 8.4).
Confirmed Commit

- `<confirmed/>` parameter in `<commit>` operation makes it a confirmed commit
  - 2nd `<commit>` required or server will rollback commit after timeout
  - Client can also send `<cancel-commit>` before the timeout
  - No warning to other clients rollback is pending
  - Client should use “persist-id” parameter to lock out other edits without explicit locks

RFC 6241, Sec. 8.4, Confirmed Commit
Confirmed Commit Exercise

- Lock datastores (!)
- Create an interface
- Confirmed commit w/ short timeout and persist-id
- Wait for timeout
- Observe rollback
- Redo but confirm the commit this time
Part 8: Edit Operations

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Edit Operations

- 3 Phase Callback Model
- Leaf-based (edit1) vs. List-based (edit2)
- Rollback
- Nested Edits
- LOAD Callback Mode
- Configuration Replay
3 Phase Database Editing Model

RPC Layer
- validate callback
- invoke callback

NETCONF server code
- edit-config
copy-config

agt_val server code
- VALIDATE
- APPLY

SIL callbacks invoked for altered nodes
- validate callback
- apply callback

COMMIT or ROLLBACK

commit or rollback callback
Edit1 Callbacks

• First Generation Edit Callbacks
  - SIL callback generated for every node (except key leafs)
  - If leaf callbacks deleted then parent container or list called N times for each leaf edited in the same request
Edit2 Callbacks

- Second Generation Edit Callbacks
  - SIL callback generated only for list, container, and choice statements
  - Parent container or list called only once for all terminal nodes edited in the same edit
    - terminal = leaf, leaf-list, anyxml
Rollback

- An error in the “validate” phase will not produce a rollback callback
- An error in the “apply” phase will produce a rollback callback
  - There will not be any commit callbacks
  - Some SIL callbacks may not get “apply” callback
- An error in the “commit” phase will produce a rollback callback
  - If commit already done for that node, then “reverse-edit” is done instead
Reverse Edit

- If commit already done for a data node and transaction is being rolled back
  - new transaction created that reverses the edit that was just committed

- New validate, apply, and commit callbacks will be done for the reverse edit
LOAD Callback Mode

• editop = OP_EDITOP_LOAD
  – Used during load from startup-cfg to running
  – Also used during configuration replay
  – Used by server to optimize validation and datastore processing
  – SIL can use this info if desired
    • same as OP_EDITOP_CREATE
Configuration Replay

• Used to resynch the main server or subsystems
  - SIL requests replay with agt_trigger_replay()
  - SIL-SA requests replay by sending a <trigger-replay> event message
• If main server not affected, then config replay is only done for subsystem(s)
  - e.g., subsystem reboots and requests replay
• No edits are recorded (or data changed)
  - Only the SIL callbacks are re-done

Developer Sec. 6.2, Configuration Replay
Edit2 Exercise

- Step through code to create and interface
  - Validate callback
  - Apply callback
  - Commit callback
Part 9: Error Handling

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
Error Handling

- `<rpc-error>` Element
- `SET_ERROR` for programming errors
- `agt_record_error` (and variants) for user errors
- `rpc_msg_add_error_data`
<rpc-error>

- `<rpc-reply>` message contains 1 or more `<rpc-error>` elements if the operation failed

```xml
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error><!-- lock failed -->
    <error-type>protocol</error-type>
    <error-tag>lock-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message>
      Lock failed, lock is already held
    </error-message>
    <error-info>
      <session-id>454</session-id>
      <!-- lock is held by NETCONF session 454 -->
    </error-info>
  </rpc-error>
</rpc-reply>
```

RFC 6241 Sec. 4.3, `<rpc-error>` Element
SET_ERROR

• This macro is for flagging programming errors
  - Do not use for normal user errors
  - E.g., an unsupported or unexpected value is used in a switch statement

```c
switch (editop) {
  case OP_EDITOP_LOAD:
    break;
  case OP_EDITOP_MERGE:
    break;
  case OP_EDITOP_REPLACE:
    break;
  case OP_EDITOP_CREATE:
    break;
  case OP_EDITOP_DELETE:
    break;
  default:
    res = SET_ERROR(ERR_INTERNAL_VAL);
}
```
agt_record_error

- Main error reporting API in the server

```c
void agt_record_error (ses_cb_t *scb,
                    xml_msg_hdr_t *msghdr,
                    ncx_layer_t layer,
                    status_t res,
                    const xml_node_t *xmlnode,
                    ncx_node_t parmtyp,
                    const void *error_info,
                    ncx_node_t nodetyp,
                    void *error_path)
```

Developer Sec. 6.3.3, agt_record_error
# agt_record_error

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scb</td>
<td>Session processing the request</td>
</tr>
<tr>
<td>msghdr</td>
<td>Message header used to store error record</td>
</tr>
<tr>
<td>layer</td>
<td>Conceptual protocol layer for the error</td>
</tr>
<tr>
<td>res</td>
<td>Internal status code for the error</td>
</tr>
<tr>
<td>xmlnode</td>
<td>XML node being parsed (may be NULL if N/A)</td>
</tr>
<tr>
<td>parmtyp</td>
<td>Type of data structure pointed at by errorinfo</td>
</tr>
<tr>
<td>errorinfo</td>
<td>Data to use for the <code>&lt;error-info&gt;</code> field</td>
</tr>
<tr>
<td>nodetyp</td>
<td>Type of data structure pointed at by error_path</td>
</tr>
<tr>
<td>error_path</td>
<td>Data to use for the <code>&lt;error-path&gt;</code> field</td>
</tr>
</tbody>
</table>
agt_record_error_errinfo

- Same as agt_record_error, plus custom errinfo

```c
void
agt_record_error_errinfo (ses_cb_t *scb,
xml_msg_hdr_t *msghdr,
ncx_layer_t layer,
status_t res,
const xml_node_t *xmlnode,
ncx_node_t parmtyp,
const void *error_info,
ncx_node_t nodetyp,
void *error_path,
const ncx_errinfo_t *errinfo)
```

Developer, Sec. 6.3.4, agt_record_error_errinfo
**ncx_errinfo_t**

- Struct used to provide custom values for some `<rpc-error>` fields
  - error-app-tag
  - error-message

```c
/* YANG error info statement struct */
typedef struct ncx_errinfo_t {
    diq_hdr_t qhdr;
    xmlChar *descr;
    xmlChar *ref;
    xmlChar *error_app_tag;
    xmlChar *error_message;
    boolean seen;           /* for yangdiff */
} ncx_errinfo_t;
```
**rpc_msg_add_error_data**

- Used to add custom data to the `<error-info>` container within the `<rpc-error>` element

```c
/* a real add_user_data callback would probably get a system
 * value instead of a constant; use constant here
 */
const xmlChar *valuebuff = (const xmlChar *)"42";

status_t res = NO_ERR;
val_value_t *testdata_val =
  val_make_simval_obj(testdata_obj, valuebuff, &res);

if (testdata_val) {
    rpc_msg_add_error_data(msg, testdata_val);
}
```

Developer Sec. 6.3.5, Adding `<error-info>`
Part 10: SIL Control

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
SIL Control

- yp-system library
- agt_util support functions
- val support functions and macros
- val_util support functions
- YANG extensions that affect SIL processing
- CLI parameters that affect SIL processing
yp-system Library

- System callbacks not related to a YANG module
  - API callbacks
    - yp_system_init_profile
    - yp_system_init1
    - yp_system_init2
    - yp_system_cleanup
  - Optional Vendor APIs
    - External Access Control Model
    - External Syslog
  - Additional Server Processing Hooks
yp-system Library
Extra Hooks

- System events that can have callbacks invoked
  - candidate reload
  - module load
  - module unload
  - commit complete
  - external NV-load handler
  - external NV-save handler
  - config-replay requested

yp-system, sec. 7, System Callback Functions
agt_util

• Error handling
  - agt_record_error
  - agt_record_error_errinfo
  - agt_record_attr_error
  - agt_record_unique_error

• agt_tree Filtering
  - agt_check_default
  - agt_check_config_false
  - agt_check_save
agt_util (2)

- Make return val_value_t subtrees
  - agt_make_leaf
  - agt_make_uint_leaf
  - agt_make_int_leaf
  - agt_make_uint64_leaf
  - agt_make_int64_leaf
  - agt_make_idref_leaf
  - agt_make_list
  - agt_make_object
  - agt_make_virtual_leaf
agt_util (3)

- Add datastore nodes
  - agt_add_top_virtual
  - agt_add_top_container
  - agt_add_container
  - agt_add_top_node_if_missing
  - agt_make_leaf2
  - agt_make_union_leaf
  - agt_make_uint_leaf2
  - agt_make_int_leaf2
  - agt_make_bits_leaf

/usr/include/yumapro/agt/agt_util.h
• **val**

  - **val**\_\_value\_t support
    - **val**\_\_new\_\_value
    - **val**\_\_init\_\_from\_\_template
    - **val**\_\_free\_\_value
    - **VAL**\_\_INT() and other macros to access value
    - **VAL**\_\_TYPE(), **VAL**\_\_NAME() macros
    - **val**\_\_simval\_\_ok and other validation functions
    - **val**\_\_dump\_\_value
    - **val**\_\_sprintf\_\_simval\_\_nc

/\usr/include/yumapro/ncx/val.h
• val_value_t support
  - val_new_value
  - val_init_from_template
  - val_free_value
  - VAL_INT() and other macros to access value
  - VAL_TYPE(), VAL_NAME() macros
  - val_simval_ok and other validation functions
  - val_dump_value
  - val_sprintf_simval_nc

/usr/include/yumapro/ncx/val.h
• Additional val_value_t support
  - val_make_simval_obj
  - val_gen_index_chain
  - val_set_canonical_order
  - val_add_defaults
  - val_instance_check
  - val_get_choice_first_set
  - val_get_choice_next_set
  - val_choice_is_set

/usr/include/yumapro/ncx/val_util.h
YANG Extensions for SIL

- nacm:default-deny-all
  - Deny all access unless explicit NACM rules
- nacm:default-deny-write
  - Deny all writes unless explicit NACM rules
- ypx:exclusive-rpc
  - Do not allow concurrent calls to the RPC
- ncx:sil-delete-children-first
  - Force child callbacks if parent deleted
- ywx:sil-force-replace-replay
  - Force replay of unchanged siblings for replace
YANG Extensions for SIL (2)

- ywx:sil-force-replay
  - Force replay of unchanged siblings for merge
- ywx:sil-priority
  - Force SIL callback order for edits
- ywx:user-write
  - Control user write access
    - override YANG config=true

Developer Sec. 12.2, Built-in Language Extension
## CLI Parameters for SIL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--bundle</td>
<td>Load a SIL or SIL-SA bundle library</td>
</tr>
<tr>
<td>--default-style</td>
<td>Set the system definition of a default leaf</td>
</tr>
<tr>
<td>--deviation</td>
<td>Load a YANG module for its deviations only</td>
</tr>
<tr>
<td>--feature-disable</td>
<td>Disable a YANG feature</td>
</tr>
<tr>
<td>--feature-enable</td>
<td>Enable a YANG feature</td>
</tr>
<tr>
<td>--feature-enable-default</td>
<td>Set YANG features default enabled or not</td>
</tr>
<tr>
<td>--log</td>
<td>Set the log file</td>
</tr>
<tr>
<td>--log-level</td>
<td>Set logging debug level</td>
</tr>
</tbody>
</table>
# CLI Parameters for SIL (2)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--no-watcher</td>
<td>Do not start YPWatcher process</td>
</tr>
<tr>
<td>--running-error</td>
<td>Allow running datastore to container validation errors</td>
</tr>
<tr>
<td>--runpath</td>
<td>Directory paths to search for libraries</td>
</tr>
<tr>
<td>--sil-missing-error</td>
<td>Force an error if SIL or SIL-SA library not found during module load</td>
</tr>
<tr>
<td>--sil-skip-load</td>
<td>Skip all OP_EDITOP_LOAD mode callbacks</td>
</tr>
</tbody>
</table>

---

netconfd-pro, Sec. 3, CLI Reference
Part 11: SIL Debugging

- Intro/Big Picture
- YANG Tutorial
- Getting Started with Server Development
- RPC Processing
- Notifications
- Retrieval Operations
- Transaction Model
- Edit Operations
- Error Handling
- SIL Control
- SIL Debugging
SIL Debugging

- Logging
- Using yangcli-pro to test the server
- sil-error
- sil-sa-app
- db-api-app
## Logging

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-level</td>
<td>Set the debug output level</td>
</tr>
<tr>
<td>log</td>
<td>Set the log file</td>
</tr>
<tr>
<td>log-append</td>
<td>Append or overwrite existing log file</td>
</tr>
<tr>
<td>log-mirror</td>
<td>Copy log output to STDOUT and log file</td>
</tr>
<tr>
<td>log-header</td>
<td>Set custom logging header</td>
</tr>
<tr>
<td>log-syslog</td>
<td>Send logging output to SYSLOG</td>
</tr>
<tr>
<td>log-backtrace</td>
<td>Add stack backtrace info to log output</td>
</tr>
<tr>
<td>log-backtrace-level</td>
<td>Set minimum logging level to add backtrace</td>
</tr>
<tr>
<td>log-event-drops</td>
<td>Log notification event drop information</td>
</tr>
<tr>
<td>log-pthread-level</td>
<td>Add PTHREAD specific logging information</td>
</tr>
</tbody>
</table>
Using yangcli-pro

- NETCONF Client Test Tool
  - variables
    - $$foo = command
    - @foo.xml = command
    - command parm=$foo
    - command parm=@foo.xml
  - scripts
    - Just text files with yangcli commands
  - test suites
    - Positive and negative response testing

YumaPro yangcli-pro Manual
- Special YANG module used to force an error during transaction processing
  - set 'sil-phase' to arm test
  - set 'sil-trigger' to any value to trigger error

```
grouping sil-error-test {
    leaf sil-phase {
        type sil-error-phase;
        description
        "Set this to the SIL callback phase you want to cause an
         operation-failed error to occur.";
    }
    leaf sil-trigger {
        type uint32;
        description
        "Set this object to trigger an operation-failed error
         if the sil-error-phase is not set to 'none'.";
    }
}
```
sil-error (2)

- YANG objects that support the sil-error objects
  - root, container, list, and choice

```
  +++rw sil-phase?    sil-error-phase
  +++rw sil-trigger?  uint32
  +++rw sil-error-con
    |  +++rw sil-phase?    sil-error-phase
    |  +++rw sil-trigger?  uint32
  +++rw sil-error-list* [id]
    |  +++rw id           string
    |  +++rw sil-phase?    sil-error-phase
    |  +++rw sil-trigger?  uint32
  +++rw sil-error-con2
    |  +++rw (sil-error-choice)?
        |    |  +++:(dummy)
        |  +++rw dummy?        int8
        |  +++:(real)
        |  +++rw sil-phase?    sil-error-phase
        |  +++rw sil-trigger?  uint32
```
sil-sa-app

- sil-sa-app is a dummy application to run SIL-SA library code
  - No special configuration needed
  - If SIL-SA library found by subsystem for configured module or bundle, then it is loaded and registered with main server
- Start main server, then run sil-sa-app
  - sil-sa-app [log-level=enum] [subsys-id=id]
db-api-app

- db-api-app is a dummy application to send a DB-API edit from a subsystem to the main server
  - DB-API used when server has an external database (e.g. CLI) and edits are initiated from the system as well as NETCONF
- Hard-wired to send 1 edit request
  - Need to edit main.c to change the edit that is sent
  - Used to demonstrate db_api_send_edit API
Thank You!