YumaPro Developer Manual

YANG-Based Unified Modular Automation Tools

Server Instrumentation Library Development

Version 21.10-4

Table of Contents

1 Preface....10
1.1 Legal Statements....10
1.2 Additional Resources....10
1.2.1 WEB Sites....10
1.2.2 Mailing Lists....11
1.3 Conventions Used in this Document....11
2 YumaPro Doxygen Browser....12
2.1.1 Online Version....12
2.1.2 Local Version....12
2.1.3 Install doxygen....13
2.1.4 Doxygen Group Structure....14
2.1.5 Example Doxygen Headers....15
3 SDK QuickStart....16
3.1 Get the YANG Modules Ready....16
3.2 Generate the Instrumentation Code Stubs....16
3.2.1 SIL....16
3.2.2 SIL Bundle....17
3.2.3 SIL-SA....18
3.2.4 SIL-SA Bundle....19
3.3 Fill in the Instrumentation Code Stubs....19
4 Software Overview....20
4.1 Introduction....21
4.1.1 Intended Audience....21
4.1.2 What does YumaPro Do?....21
4.1.3 What is a YumaPro Root?....22
4.1.4 Searching YumaPro Roots....23
4.1.5 What is a SIL?....24
4.1.6 SIL Variants....25
4.1.7 Auto-generated SIL Files....27
4.1.8 Basic Development Steps....29
4.2 YumaPro Source Files....30
4.2.1 src/ncx Directory....31
4.2.2 src/platform Directory....35
4.2.3 src/agt Directory....36
4.2.4 src/mgr Directory....40
4.2.5 src/subsys-pro Directory....41
4.2.6 src/netconfd-pro Directory....41
4.2.7 src/yangcli-pro Directory....41
4.2.8 src/yangdiff-pro Directory....41
4.2.9 src/yangdump-pro Directory....41
4.2.10 src/yangdump-sdk Directory....42
4.2.11 src/ydump Directory....42
4.2.12 src/ypwatcher Directory....43
4.2.13 src/ypgnmi Directory....43
4.2.14 src/ypgrpc Directory....43
4.3 Server Design....45
4.3.1 YANG Native Operation....46

4.3.2 YANG Object Tree....47
4.3.3 YANG Data Tree....48
4.3.4 Service Layering....49
4.3.5 Session Control Block....51
4.3.6 Server Message Flows....51
4.3.7 Main ncxserver Loop....53
4.3.8 SIL Callback Functions....55
4.4 Server Initialization....56
4.4.1 Set the Server Profile....56
4.4.2 Bootstrap CLI and Load Core YANG Modules....57
4.4.3 External System Init Phase I : Pre CLI....58
4.4.4 Load CLI Parameters....59
4.4.5 Load Main Configuration File....59
4.4.6 Load Secondary Configuration Files....59
4.4.7 Configuration Parameter Activation....59
4.4.8 External System Init Phase I : Post CLI....60
4.4.9 Load SIL Libraries and Invoke Phase I Callbacks....60
4.4.10 External System Init Phase II : Pre-Load....60
4.4.11 Load <running> Configuration....61
4.4.12 Invoke SIL Phase II Callbacks....61
4.4.13 External System Init Phase II : Post-Load....61
4.4.14 Initialize SIL-SA Subsystems....61
4.4.15 Server Ready....62
4.5 Server Operation....62
4.5.1 Ypwatcher processing....62
4.5.2 Loading Modules and SIL Code....63
4.5.3 Core Module Initialization....64
4.5.4 Startup Configuration Processing....64
4.5.5 Process an Incoming <rpc> Request....65
4.5.6 Edit the Database....66
4.5.7 Save the Database....67
4.6 Built-in Server Modules....68
4.6.1 iana-crypt-hash.yang....68
4.6.2 ietf-datastores.yang....68
4.6.3 ietf-inet-types.yang....68
4.6.4 ietf-origin.yang....68
4.6.5 ietf-netconf.yang....68
4.6.6 ietf-netconf-acm.yang....68
4.6.7 ietf-netconf-nmda.yang....69
4.6.8 ietf-netconf-monitoring.yang....69
4.6.9 ietf-netconf-notifications.yang....69
4.6.10 ietf-netconf-partial-lock.yang....69
4.6.11 ietf-netconf-with-defaults.yang....69
4.6.12 ietf-restconf.yang....69
4.6.13 ietf-restconf-monitoring.yang....70
4.6.14 ietf-yang-library.yang....70
4.6.15 ietf-yang-patch.yang....70
4.6.16 ietf-yang-types.yang....70
4.6.17 nc-notifications.yang....70
4.6.18 netconfd-pro.yang....70
4.6.19 notifications.yang....71
4.6.20 yang-data-ext.yang....71

4.6.21 yuma-app-common.yang....71
4.6.22 yuma-ncx.yang....71
4.6.23 yuma-mysession.yang....71
4.6.24 yuma-netconf.yang....72
4.6.25 yuma-system.yang....72
4.6.26 yuma-time-filter.yang....72
4.6.27 yuma-types.yang....72
4.6.28 yumaworks-agt-profile.yang....72
4.6.29 yumaworks-app-common.yang....72
4.6.30 yumaworks-attrs.yang....73
4.6.31 yumaworks-config-change.yang....73
4.6.32 yumaworks-db-api.yang....73
4.6.33 yumaworks-event-filter.yang....73
4.6.34 yumaworks-extensions.yang....73
4.6.35 yumaworks-getbulk.yang....73
4.6.36 yumaworks-ids.yang....73
4.6.37 yumaworks-internal.yang....74
4.6.38 yumaworks-mgr-common.yang....74
4.6.39 yumaworks-opsmgr.yang....74
4.6.40 yumaworks-restconf-commit.yang....74
4.6.41 yumaworks-restconf.yang....74
4.6.42 yumaworks-server.yang....74
4.6.43 yumaworks-sesmgr.yang....74
4.6.44 yumaworks-sil-sa.yang....75
4.6.45 yumaworks-support-save.yang....75
4.6.46 yumaworks-system.yang....75
4.6.47 yumaworks-templates.yang....75
4.6.48 yumaworks-term-msg.yang....75
4.6.49 yumaworks-test.yang....75
4.6.50 yumaworks-types.yang....75
4.6.51 yumaworks-yang-api.yang....76
4.6.52 yumaworks-yangmap.yang....76
4.6.53 yumaworks-ycontrol.yang....76
4.6.54 yumaworks-yp-gnmi.yang....76
4.6.55 yumaworks-yp-grpc.yang....76
4.6.56 yumaworks-grpc-mon.yang....76
4.6.57 yumaworks-yp-ha.yang....76
4.7 Optional Server Modules....77
4.7.1 yuma-arp.yang....77
4.7.2 yuma-proc.yang....77
4.7.3 yuma-interfaces.yang....77
4.7.4 ietf-restconf-monitoring.yang....77
5 YANG Objects and Data Nodes....78
5.1 Object Definition Tree....78
5.1.1 Object Node Types....78
5.1.2 Object Node Template (obj_template_t)....80
5.1.3 obj_template_t Access Functions....82
5.2 Data Tree....88
5.2.1 Child Data Nodes....89
5.2.2 Data Node Types....91
5.2.3 YumaPro Data Node Edit Variables (val_editvars_t)....93
5.2.4 YumaPro Data Nodes (val_value_t)....95

5.2.5 YumaPro Data Nodes (Extra) (val_extra_t)....99
5.2.6 val_value_t Access Macros....101
5.2.7 val_value_t Access Functions....103
5.2.8 SIL Utility Functions....110
6 SIL External Interface....112
6.1 Stage 1 Initialization....112
6.2 Stage 2 Initialization....115
6.3 Cleanup....117
7 SIL Callback Interface....118
7.1 Configuration Initialization....119
7.1.1 Validation Phase....120
7.1.2 Apply Phase....121
7.1.3 Commit/Rollback Phase....122
7.2 Configuration Replay....123
7.2.1 Timer Callback to Check System....123
7.2.2 SIL Callbacks....125
7.2.3 Configuration Replay Callbacks....127
7.2.4 Configuration Replay Cookie....127
7.3 Error Handling....128
7.3.1 <rpc-error> Contents....130
7.3.2 SET_ERROR....132
7.3.3 agt_record_error....133
7.3.4 agt_record_error_errinfo....135
7.3.5 agt_record_warning....136
7.3.6 Adding <error-info> Data to an Error Response....137
7.4 RPC Operation Interface....142
7.4.1 RPC Callback Template....142
7.4.2 RPC Callback Initialization....143
7.4.3 RPC Message Header....144
7.4.4 SIL Support Functions For RPC Operations....147
7.4.5 RPC Validate Callback Function....148
7.4.6 RPC Invoke Callback Function....152
7.4.7 RPC Data Output Handling....155
7.4.8 RPC Post Reply Callback Function....158
7.5 YANG 1.1 Action Interface....160
7.5.1 Action Callback Template....162
7.5.2 Action Callback Initialization....163
7.5.3 Action Message Header....163
7.5.4 SIL Support Functions For YANG Actions....163
7.5.5 Action Validate Callback Function....164
7.5.6 Action Invoke Callback Function....166
7.5.7 Action Data Output Handling....169
7.6 Database Operations....170
7.6.1 EDIT2 Callbacks....171
7.6.2 SIL-SA EDIT2 Callbacks....171
7.6.3 Database Template (cfg_template_t)....172
7.6.4 Database Access Functions....174
7.6.5 Database Callback Initialization and Cleanup....176
7.6.6 Example SIL Database Edit Callback Function....179
7.6.7 EDIT2 Callback Function Example....181
7.6.8 SIL-SA EDIT2 Callback Function Example....184

7.6.9 Database Edit Validate Callback Phase....187
7.6.10 Database Edit Apply Callback Phase....187
7.6.11 Database Edit Commit Callback Phase....188
7.6.12 Database Edit Rollback Callback Phase....188
7.6.13 Database Virtual Node Get Callback Function....188
7.6.14 Getting the Current Transaction ID....191
7.6.15 Finding Data Nodes with XPath....192
7.6.16 Determining if a Value Node is Set....195
7.6.17 Determining if the SIL Callback is the Deepest for the Edit....195
7.7 Database Editing with In-Transaction APIs....197
7.7.1 Set-Hook Callback....200
7.7.2 Set Hook Callback Examples....201
7.7.3 SIL-SA Set-Hook Callback....206
7.7.4 SIL-SA Set Hook Callback Examples....207
7.7.5 Post Set Hook Callback....212
7.7.6 SIL-SA Post Set Hook Callback....213
7.7.7 Transaction Hook Callback....215
7.7.8 Transaction Hook Callback Examples....216
7.7.9 SIL-SA Transaction Hook Callback....218
7.7.10 SIL-SA Transaction Hook Callback Examples....219
7.7.11 Transaction Start Callback....222
7.7.12 Transaction Start Callback Examples....223
7.7.13 SIL-SA Transaction Start Callback....224
7.7.14 SIL-SA Transaction Start Callback Examples....225
7.7.15 Transaction Complete Callback....227
7.7.16 Transaction Complete Callback Examples....228
7.7.17 SIL-SA Transaction Complete Callback....229
7.7.18 SIL-SA Transaction Complete Callback Examples....230
7.7.19 Set Order Hook Callback....231
7.7.20 Set Order Hook Callback Examples....232
7.7.21 Add Edit API....235
7.7.22 Add Edit Extended API....236
7.7.23 Add Edit Maximum API....238
7.7.24 SIL-SA Add Edit API....240
7.7.25 Get Data API....242
7.7.26 SIL-SA Get Data API....243
7.7.27 Startup Hook Callback....244
7.7.28 Startup Hook Callback Example....245
7.7.29 Validate Complete Callback....246
7.7.30 Validate Complete Callback Examples....247
7.7.31 SIL-SA Validate Complete Callback....249
7.7.32 SIL-SA Validate Complete Callback Examples....250
7.7.33 Apply Complete Callback....252
7.7.34 Apply Complete Callback Examples....253
7.7.35 SIL-SA Apply Complete Callback....255
7.7.36 SIL-SA Apply Complete Callback Examples....256
7.7.37 Commit Complete Callback....258
7.7.38 Commit Complete Callback Examples....259
7.7.39 SIL-SA Commit Complete Callback....261
7.7.40 SIL-SA Commit Complete Callback Examples....262
7.7.41 Rollback Complete Callback....264
7.7.42 SIL-SA Rollback Complete Callback....265

7.7.43 SIL-SA Rollback Complete Callback Examples....266
7.7.44 Dynamic Default Hook Callback....268
7.8 Notifications....270
7.8.1 Notification Send Function....270
7.8.2 Notification Send Function Variants....271
7.8.3 Event Stream Callback Functions....273
7.9 Operational Data....273
7.9.1 Get2 Object Template Based Operational Data....273
7.9.2 GETBULK Support....285
7.9.3 Example get2 callbacks....286
7.9.4 All In One GET2 Callback....291
7.9.5 AIO List Example....293
7.9.6 AIO Container Example....298
7.9.7 All In One GET2 Callbacks with XML/JSON buffers....304
7.9.8 Static Operational Data....310
Example Static Node Creation....310
7.9.9 Virtual Operational Data....311
Get Callback Template....311
Get Callback Example....311
7.10 Periodic Timer Service....313
7.10.1 Timer Callback Function....313
7.10.2 Timer Access Functions....314
7.10.3 Example Timer Callback Function....314
7.11 Terminal Diagnostic Messages....315
8 Network Management Datastore Architecture (NMDA)....317
8.1 NMDA Support....317
8.2 Instrumentation Changes for NMDA....317
8.3 Example NMDA GET2 Function....318
9 Development Environment....322
9.1 Programs and Libraries Needed....322
9.2 SIL Shared Libraries....324
9.2.1 SIL Library Names....325
9.2.2 SIL Library Location....325
9.2.3 make_sil_dir_pro....326
9.2.4 make_sil_bundle....327
9.2.5 Building SIL Libraries....327
9.3 SIL-SA Libraries....328
9.3.1 SIL-SA Library Names....329
9.3.2 SIL-SA Library Location....329
9.3.3 make_sil_sa_dir....330
9.4 SIL-SA Bundles....331
9.4.1 make_sil_sa_bundle....331
9.5 Static SIL-SA Libraries....332
9.5.1 Static SIL-SA Library Example....332
9.6 SIL Makefile....334
9.6.1 Target Platforms....334
9.6.2 Build Targets....334
9.6.3 Command Line Build Options....335
9.6.4 Example SIL Makefile....335
9.7 Controlling SIL Behavior....336

9.7.1 SIL Invocation Order (sil-priority)....336
9.7.2 Reverse SIL priority for delete operations (--sil-prio-reverse-for-deletes)....337
9.7.3 Deletion of Child Nodes (sil-delete-children-first)....338
9.7.4 Suppress leafref Validation....339
9.8 SIL-SA APIs....340
9.8.1 Access MSG Values....341
9.8.2 SIL-SA EDIT APIs....341
9.9 Short Names for SIL and SIL-SA Code Generation....344
9.9.1 CLI Parameters....344
9.9.2 Short Name Summary....345
9.9.3 Constants....346
9.9.4 Function and Variable Names....348
10 YControl Subsystem....349
10.1 YControl API Functions....351
10.1.1 Initialization and Cleanup....351
10.1.2 Runtime Functions....353
10.2 YControl Message Structure....355
10.2.1 yumaworks-ycontrol YANG Module....356
11 SIL-SA Subsystem....360
11.1 Example SIL-SA Application....360
11.2 SIL-SA Message Format....363
11.2.1 SIL-SA Registration Message Flow....363
11.2.2 Server Edit Transaction Message Flow....364
11.2.3 SIL-SA Register Request Message....365
11.2.4 SIL-SA Start Transaction Message....366
11.2.5 SIL-SA Continue Transaction Message....367
11.2.6 SIL-SA Transaction Response Message....368
11.2.7 SIL-SA Cancel Transaction Event....369
11.2.8 SIL-SA Load Event....370
11.2.9 SIL-SA Bundle Load Event....371
11.2.10 SIL-SA Get Request Message....372
11.2.11 SIL-SA Get Response Message....373
11.2.12 SIL-SA Notification Message....374
11.2.13 SIL-SA RPC Request Message....375
11.2.14 SIL-SA RPC Response Message....376
11.2.15 SIL-SA Action Request Message....377
11.2.16 SIL-SA Action Response Message....378
11.2.17 SIL-SA Transaction Start Message....379
11.2.18 SIL-SA Transaction Complete Event....380
11.2.19 SIL-SA Hook Get Request Message....381
11.2.20 SIL-SA Hook Get Response Message....382
11.2.21 SIL-SA Stream Callback Event....383
11.2.22 SIL-SA Commit Completeness Hook Message....384
11.2.23 yumaworks-sil-sa YANG Module....385
12 DB-API Subsystem....408
12.1 DB-API Interface Functions....415
12.1.1 Filtered Retrieval Example....418
12.1.2 Subsystem RPC Request Example....419
12.1.3 Subsystem Action Request Example....421
12.2 DB-API Messages....423
13 YumaPro gNMI Subsystem....430

13.1 gNMI Client to Netconfd-pro Processing....432
13.1.1 Ypgnmi-app Processing....432
13.2 Ypgnmi-app Message Format....434
13.2.1 Ypgnmi-app Registration Message Flow....435
13.2.2 Yumaworks-yp-gnmi YANG Module....436
13.3 Netconfd-pro Processing....447
13.4 gNMI Service definition....448
13.4.1 gNMI GetRequest....449
13.4.2 gNMI SetRequest....452
13.4.3 gNMI JSON_ietf_val....454
13.4.4 gNMI Error Messages....454
14 YumaPro gRPC Subsystem....455
14.1 Ypgrpc-go-app Overview....456
14.1.1 Ypgrpc-go-app Benefits....456
14.1.2 Ypgrpc-go-app Processing....456
14.1.3 Startup Procedure....457
14.1.4 Running Ypgrpc-go-app....458
14.1.5 Closing Ypgrpc-go-app....458
14.2 Ypgrpc-go-app Message Format....459
14.2.1 YControl Integration....459
14.2.2 Ypgrpc-go-app Registration Message Flow....460
14.2.3 Yumaworks-yp-grpc YANG Module....461
15 Automation Control....465
15.1 YANG Parser Object Template APIs....465
15.1.1 Object Template User Flags....465
15.1.2 Object Template Callback API....466
15.1.3 YANG Object template Callback Usage Example....467
15.2 YANG Parser Extension Statement APIs....469
15.2.1 YANG Extension Handler Callback....469
15.2.2 Usage Example....471
15.3 Abstract YANG Data APIs....473
15.3.1 rc:yang-data....473
15.3.2 yd:augment-yang-data....473
15.4 SIL Language Extension Access Functions....474
15.5 Built-in YANG Language Extensions....474
15.5.1 ncx:abstract....475
15.5.2 ywx:alt-name....475
15.5.3 ncx:cli....476
15.5.4 ywx:cli-text-block....477
15.5.5 ywx:datapath....478
15.5.6 nacm:default-deny-all....479
15.5.7 nacm:default-deny-write....479
15.5.8 ncx:default-parm....480
15.5.9 ncx:default-parm-equals-ok....481
15.5.10 ywx:get-filter-element-attributes....481
15.5.11 ywx:exclusive-rpc....482
15.5.12 ywx:help....482
15.5.13 ncx:hidden....483
15.5.14 ncx:metadata....483
15.5.15 ncx:no-duplicates....484
15.5.16 ncx:password....484

15.5.17 ncx:qname....485
15.5.18 oc-ext:openconfig-hashed-value....485
15.5.19 oc-ext:regexp-posix....487
15.5.20 ncx:root....488
15.5.21 ywx:rpc-root....488
15.5.22 ncx:schema-instance....489
15.5.23 nacm:secure....490
15.5.24 ncx:sil-aio-get2....491
15.5.25 ncx:sil-delete-children-first....492
15.5.26 ywx:sil-force-replace-replay....493
15.5.27 ywx:sil-force-replay....494
15.5.28 ywx:sil-priority....495
15.5.29 ywx:sil-test-get-when....496
15.5.30 ywx:urlpath....497
15.5.31 ncx:user-write....498
15.5.32 nacm:very-secure....499
15.5.33 ncx:xpath....500
15.5.34 ywx:xpath-operational-ok....501
15.5.35 ncx:xsdlist....502

1 Preface

1.1 Legal Statements

1.2 Additional Resources

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

YumaPro Installation Guide

YumaPro Quickstart Guide

Other documentation includes:

YumaPro API Quickstart Guide

YumaPro Quickstart Guide

YumaPro User Manual

YumaPro netconfd-pro Manual

YumaPro yangcli-pro Manual

YumaPro yangdiff-pro Manual

YumaPro yangdump-pro Manual

YumaPro ypclient-pro Manual

YumaPro ypgnmi Manual

YumaPro ypgrpc Manual

YumaPro yp-system API Guide

YumaPro yp-show API Guide

YumaPro Yocto Linux Quickstart Guide

YumaPro yp-snmp Manual

To obtain additional support contact YumaWorks technical support:

support@yumaworks.com

1.2.1 WEB Sites

- **YumaWorks**
  - https://www.yumaworks.com
  - Offers support, training, and consulting for YumaPro.
- **Netconf Central**
  - http://www.netconfcentral.org/

  - Free information on NETCONF and YANG, tutorials, on-line YANG module validation and documentation database

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

1.2.2 Mailing Lists

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

1.3 Conventions Used in this Document

The following formatting conventions are used throughout this document:

Documentation Conventions

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

2 YumaPro Doxygen Browser

The YumaPro server code now supports doxygen. The H files in the 20.10 release train have been redone so they conform to the appropriate doxygen and markdown commands. The HTML generation is supported.

The doxygen output is available online and can also be generated on a local machine if an SDK package or source code package is installed.

The SIL and SIL-SA code generated by yangdump-sdk now has built-in doxygen browser support.

It is strongly recommended that the browser be used to learn the APIs and access additional technical support resources.

The doxygen program is a widely available open-source program for generating source code documentation.

- Doxygen Home Page:
  - https://www.doxygen.nl/index.html

2.1.1 Online Version

The output for the latest YumaPro release is available online

- YumaPro Doxygen Browser
  - https://www.yumaworks.com/pub/doxygen/latest/html/index.html

2.1.2 Local Version

If the source code or a binary SDK package is installed, then the doxygen browser can be built and viewed is a browser as a file. The following scripts now support doxygen output.

- make_sil_dir_pro
- make_sil_bundle
- make_sil_sa_dir
- make_sil_sa_bundle

The following message is printed after the script runs, showing the steps needed to access the local doxygen browser. (E.g. module

```
> make_sil_dir_pro --split test --sil-get2 --sil-edit2

  .  .  .

Run the following commands to get started:

  cd test
  make doc
  make opendoc

>
```

The doxygen files can be generated after the script is run.

The URL can also be access locally. The “index.html” file will be in the directory “output/html” under the module root created (E.g. “test” in this example).

2.1.3 Install doxygen

This step is not required to generate any server source code. It is only required to use doxygen.

The doxygen and graphviz packages are used to generate all the doxygen documentation, from your source code and the installed YumaPro H files.

These programs are usually already installed, but if not then try these commands:

```
Ubuntu:

   sudo apt-get install doxygen graphviz

Fedora:

   sudo dnf install doxygen graphviz

Centos:

   sudo dnf --enablerepo=powertools install doxygen graphviz
```

2.1.4 Doxygen Group Structure

Doxygen uses a simple hierarchy to create the 'Modules' section of the WEB pages for the source code.

- This hierarchy is hard-wired at this time.
- The top-level group is called "yang-library" and the brief description is simply "YANG Library".
  - The "ingroup" command for the auto-generated code will use this value
- The 2nd-level group depends on the code that is being generated. There are parts to the group name
  - "sil-" or "silsa-" depending on code for SIL or SIL-SA
  - module name or bundle name
- If the generated source file is for a module within a bundle, then there will be a 3rd level of grouping.
  - Each module will have a grouping within the bundle grouping

Example tree for SIL code for "module1"

```
+-- yang-library
    +
    |
    +-- sil-module1
```

Example tree for SIL-SA code for "bundle1"

```
+-- yang-library
    +
    |
    +-- silsa-bundle1
```

Example tree for SIL-SA code for "module1" and "module2" within "bundle1"

```
+-- yang-library
    +
    |
    +-- silsa-bundle1
        +
        |
        + silsa-bundle1-module1
        + silsa-bundle1-module2
```

2.1.5 Example Doxygen Headers

New GET2 callback for interface statistics:

```
/**
 * @brief Get database object callback for container statistics (getcb_fn2_t)\n
 * Path: container /interfaces/interface/statistics\n
 *
 * Fill in 'get2cb' response fields.
 *
 * @param get2cb GET2 control block for the callback.
 * @param k_if_name Ancestor key leaf 'name' in list 'interface'\n
 * Path: /if:interfaces/if:interface/if:name
 * @return return status of the callback.
 */
extern status_t u_if_statistics_get (
    getcb_get2_t *get2cb,
    const xmlChar *k_if_name);
```

New EDIT2 callback for interface:

```
/**
 * @brief Edit database object callback (agt_cb_fn_t)\n
 * Path: list /interfaces/interface
 *
 * @param scb session control block making the request
 * @param msg message in progress for this edit request
 * @param cbtyp callback type for this callback
 * @param editop the parent edit-config operation type,
 * which is also used for all other callbacks
 * that operate on objects.
 * @param newval container object holding the proposed changes
 * to apply to the current config, depending on
 * the editop value. Will not be NULL.
 * @param curval current container values from the "<running>"
 * or "<candidate>" configuration, if any. Could be NULL
 * for create and other operations.
 * @param k_if_name Local key leaf 'name' in list 'interface'\n
 * Path: /if:interfaces/if:interface/if:name
 * @return return status for the phase.
 */
extern status_t u_if_interface_edit (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    agt_cbtyp_t cbtyp,
    op_editop_t editop,
    val_value_t *newval,
    val_value_t *curval,
    const xmlChar *k_if_name);
```

3 SDK QuickStart

This section gives a quick overview for those who want to get started with the netconfd-pro server without reading the entire manual first. It is assumed that the appropriate programs have been installed and configured correctly.

3.1 Get the YANG Modules Ready

- Make sure that the correct versions of the YANG modules needed for your development are visible to the YANG compiler and the server. The --**modpath** CLI variable and **YUMAPRO_MODPATH** environment variables can be used to specify a non-default search order.
- Add any YANG extension statements to the module needed to control server behavior. These extensions are described in the Automation Control section.

3.2 Generate the Instrumentation Code Stubs

The server instrumentation code stubs are used to hook YANG data models to system instrumentation.

3.2.1 SIL

- The SIL code template is used for instrumentation included in the main server (not a sub-agent).
- All YANG data nodes, remote procedure call (RPC) operations, and notification event instrumentation code can be located in a SIL library.
- Use the **make_sil_dir_pro** script to generate the SIL code stubs for a single YANG module
- Example: **test.yang**

```
> make_sil_dir_pro --split test --sil-get2 --sil-edit2
```

- The --**split** parameter is now strongly recommended. A deprecation warning will be generated if this parameter is not used. It generates more user-friendly "**u_"** user files that you edit. It also generates "**y_**" YumaPro files that you should never need to edit.
- The --**sil-edit2** and --**sil-get2** parameters select the current GET2 and EDIT2 code generation modes.
- The C and H files will be generated in the **./test/src** directory
- Run "make" and "sudo make install" from the SIL directory (E.g., "test"). There will be some warnings but no errors.
- The SIL library file (E.g. **libtest.so**) will be installed in the YumaPro library directory (usually **/usr/lib/yumapro**).
- The SIL library will be loaded dynamically by the server (**--module=test** or **load test)**

3.2.2 SIL Bundle

- The SIL Bundle code template is used for instrumentation included in the main server (not a sub-agent).
- All YANG data nodes, remote procedure call (RPC) operations, and notification event instrumentation code can be located in a SIL bundle library.
- Use the **make_sil_bundle** script to generate the SIL bundle code stubs for multiple YANG modules at once.
- This mode should be used when external modules augment a base module (such as i**etf-interfaces)**
- Example: **if-bundle**

```
> make_sil_bundle if-bundle ietf-interfaces acme-interface-extensions \
   --sil-get2 --sil-edit2
```

- This example will generate a bundle named “if-bundle”, that contains the “ietf-interfaces” and “acme-interface-extensions” modules.
- The **--sil-edit2** and **--sil-get2** parameters select the current GET2 and EDIT2 code generation modes.
- The C and H files will be generated in the **./test/src** directory. Extra C and H files will be created for the bundle itself. These are used for initialization and cleanup only.
- Run “make” and “sudo make install” from the SIL bundle directory (E.g., “if-bundle”). There will be some warnings but no errors.
- The SIL bundle library file (E.g. **libif-bundle.so**) will be installed in the YumaPro library directory (usually **/usr/lib/yumapro**).
- The SIL bundle library will be loaded dynamically by the server (**--bundle=if-bundle** or **load-bundle if-bundle)**

3.2.3 SIL-SA

- The SIL-SA code template is used for instrumentation included in a sub-agent (not the main server).
- All YANG data node instrumentation code can be located in a SIL-SA library.
- Use the **make_sil_sa_dir** script to generate the SIL-SA code stubs for a single YANG module
- Example: **test.yang**

```
> make_sil_sa_dir --split test --sil-get2 --sil-edit2
```

- The --**split** parameter is recommended. It generates more user-friendly "**u_"** user files that you edit. It also generates "**y_**" YumaPro files that you should never need to edit.
- The **--sil-edit2** and **--sil-get2** parameters select the current GET2 and EDIT2 code generation modes.
- The C and H files will be generated in the **./test/src** directory
- Run "make" and "sudo make install" from the SIL-SA directory (E.g., "test"). There will be some warnings but no errors.
- The SIL-SA library file (E.g. **libtest_sa.so**) will be installed in the YumaPro library directory (usually **/usr/lib/yumapro**). Note the extra "_sa" string in the library name.
- The SIL-SA library will be loaded dynamically by the server (**--module=test** or **load test)**
- The SIL-SA library for a module can be built as a static or a dynamic library, that can be linked into the **sil-sa-app** or **combo-app** program template.

3.2.4 SIL-SA Bundle

- The SIL-SA Bundle code template is used for instrumentation included in a sub-agent (not the main server).
- All YANG data node instrumentation code can be located in a SIL-SA library.
- Use the **make_sil_sa_bundle** script to generate the SIL-SA bundle code stubs for multiple YANG modules at once.
- This mode should be used when external modules augment a base module (such as i**etf-interfaces)**
- Example: **if-bundle**

```
> make_sil_sa_bundle if-bundle ietf-interfaces iana-if-type \
  acme-interface-extensions --sil-get2 --sil-edit2
```

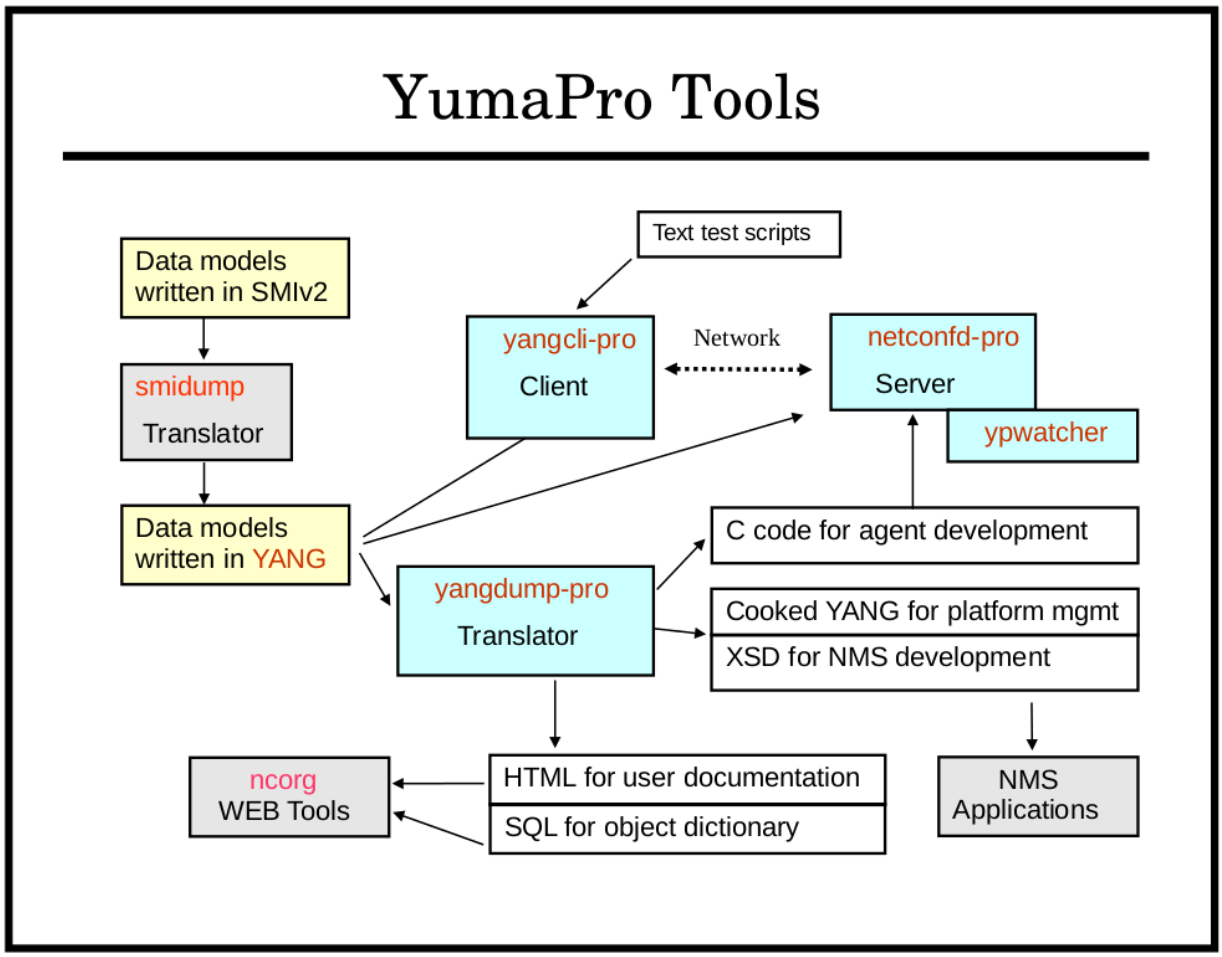
- This example will generate a bundle named “if-bundle”, that contains the “ietf-interfaces” and “acme-interface-extensions” modules.
- The **--sil-edit2** and **--sil-get2** parameters select the current GET2 and EDIT2 code generation modes.
- The C and H files will be generated in the **./test/src** directory. Extra C and H files will be created for the bundle itself. These are used for initialization and cleanup only.
- Run “make” and “sudo make install” from the SIL-SA bundle directory (E.g., “if-bundle”). There will be some warnings but no errors.
- The SIL-SA bundle library file (E.g, **libif-bundle_sa.so**) will be installed in the YumaPro library directory (usually **/usr/lib/yumapro**). Note the extra “_sa” string in the library name.
- The SIL-SA bundle library will be loaded dynamically by the server (**--bundle=if-bundle** or **load-bundle if-bundle)**
- The SIL-SA library for a bundle can be built as a static or a dynamic library, that can be linked into the **sil-sa-app** or **combo-app** program template.

3.3 Fill in the Instrumentation Code Stubs

The code stubs will contain comments indicating where you need to add instrumentation code to process a configuration edit or retrieval request. Refer to the SIL Callback Interface section on the details of these functions.

Generally, the **val_value_t** parameter “newval” contains the new value for an edit and the parameter “curval” contains the current value. This will be NULL if the node is being created. The macros in **ncx/val.c** (E.g, VAL_UINT32(val)) are used to access these parameters.

4 Software Overview



4.1 Introduction

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

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

4.1.1 Intended Audience

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

4.1.2 What does YumaPro Do?

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

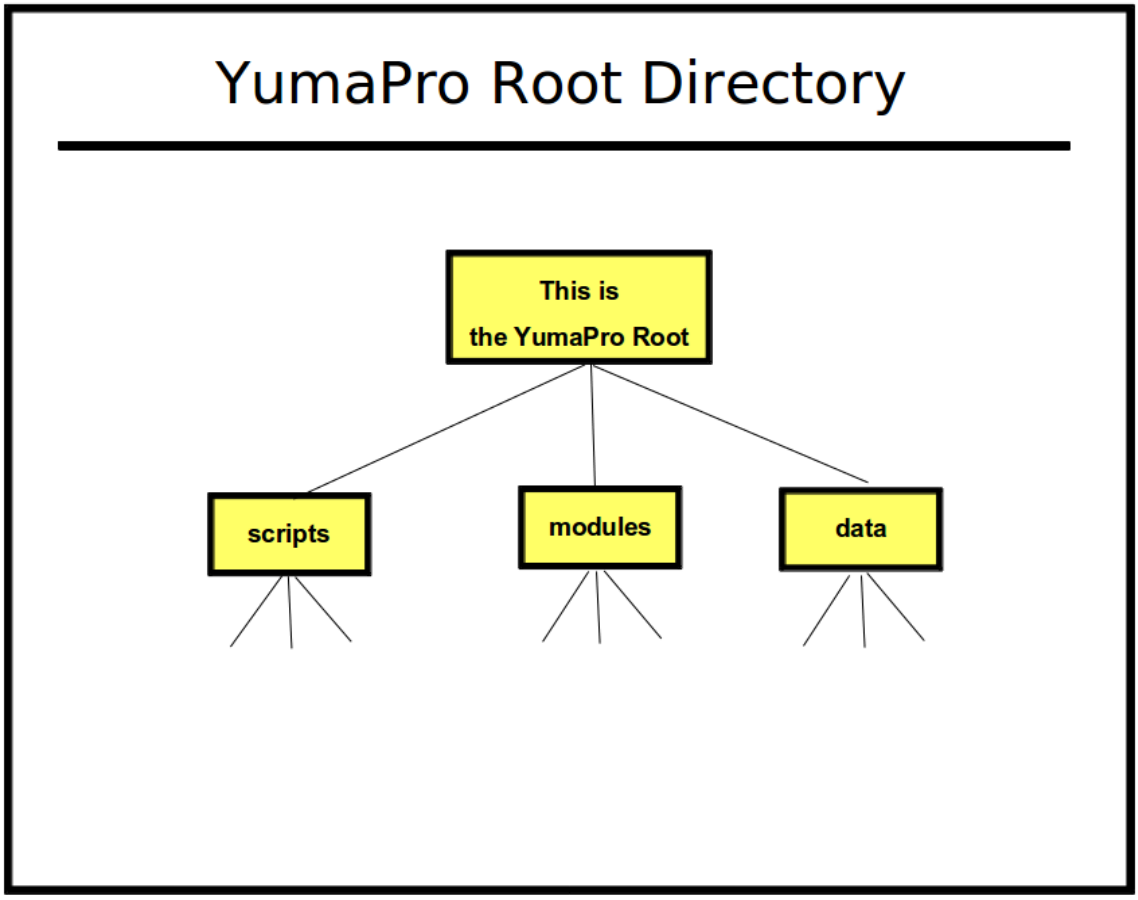
This section describes the YumaPro development environment and the basic tasks that a software developer needs to perform, in order to integrate YANG module instrumentation into a device.

This manual contains the following information:

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

YumaPro 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.

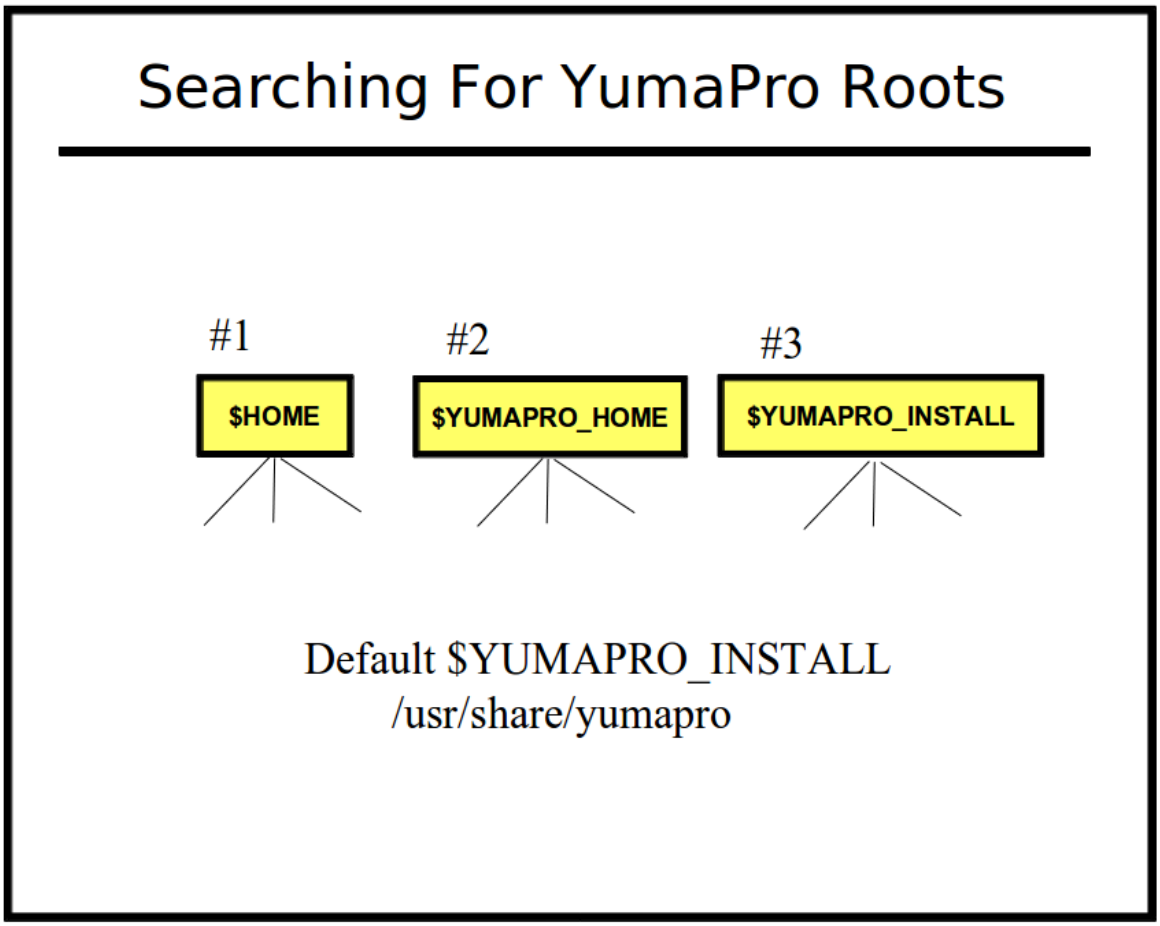
4.1.3 What is a YumaPro Root?



The YumaPro Tools programs will search for some types of files in default locations

- **YANG Modules**: The 'modules' sub-directory is used as the root of the YANG module library.
- **Client Scripts**: The **yangcli-pro** program looks in the 'scripts' sub-directory for user scripts.
- **Program Data**: The **yangcli-pro** and netconfd-pro programs look for saved data structures in the 'data' sub-directory.

4.1.4 Searching YumaPro Roots



1) $HOME Directory

The first YumaPro root checked when searching for files is the directory identified by the $HOME environment variable. If a '**$HOME/modules**', '**$HOME/data**'. and/or '**$HOME/scripts**' directory exists, then it will be checked for the specified file(s).

2) The $YUMAPRO_HOME Directory

The second YumaPro root checked when searching for files is the directory identified by the $YUMAPRO_HOME environment variable. This is usually set to private work directory, but a shared directory could be used as well. If a '**$YUMAPRO_HOME/modules**', '**$YUMAPRO_HOME/data**'. and/or '**$YUMAPRO_HOME/scripts**' directory exists, then it will be checked for the specified file(s).

3) The $YUMAPRO_INSTALL Directory

The last YumaPro root checked when searching for files is the directory identified by the **$YUMAPRO_INSTALL** environment variable. If it is not set, then the default value of '**/usr/share/yumapro**' is used instead. This is usually set to the public directory where all users should find the default modules. If a '**$YUMAPRO_INSTALL/modules**', '**$YUMAPRO_INSTALL/data**'. and/or '**$YUMAPRO_INSTALL/scripts**' directory exists, then it will be checked for the specified file(s).

4.1.5 What is a SIL?

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

The **netconfd-pro** 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-pro** 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-pro** server. The 'load' command (via CLI, configuration file, protocol operation) is used (by the operator) to activate the YANG module and its SIL.

4.1.6 SIL Variants

Server instrumentation libraries can be local or remote:

- SIL: synchronous local server instrumentation library within the netconfd-pro process via callback functions
- SIL-SA: asynchronous remote sub-agent server instrumentation library, within another process, connected via YControl messages

A SIL or SIL-SA can contain the instrumentation for one module or several modules (called a bundle).

It is strongly recommended that a bundle be used for a base module and the modules that augment that base module. This allows the callback code for augmenting modules to be generated correctly by **yangdump-sdk**. Any modules can be included a bundle, not just a base modules and modules that augment it.

It is strongly recommended that the **--sil-get2** and **--sil-edit2** parameters be used for all SIL code.

There are 4 scripts that can be used to generate SIL or SIL-SA code stubs

make_sil script

Script	Description
make_sil_dir_pro	Generate C code stubs for a combined or split SIL variant
make_sil_sa_dir	Generate C code stubs for a combined or split SIL-SA variant
make_sil_bundle	Generate C code stubs for a split SIL bundle variant
make_sil_sa_bundle	Generate C code stubs for a split SIL-SA bundle variant

There are some script parameters that allow different variants of the C code stubs to be generated

SIL Variants

Variant	Param	Description
SIL		SIL or SIL-SA for 1 module
BUNDLE	--sil-bundle	SIL or SIL-SA for 1 bundle
COMBINED		Y and U instrumentation combined
SPLIT	--split	Y and U instrumentation separate (recommended)

Extra Variant Parameters

Variant	Param	Description
GET2	--sil-get2	generate 2nd generation operational data callbacks (recommended)
EDIT2	--sil-edit2	generate 2nd generation edit callbacks (recommended)

There are some **yangdump-sdk** parameters that can be used with the **make_sil** scripts and they will be passed to the **yangdump-sdk** program

yangdump-sdk Parameters

Param	Description
--indent	Set the code indent level (script default 4)
--deviation	Process the specified YANG deviation so SIL code will be generated for deviated module
-sil-include	Specify an #include directive to be generated

4.1.7 Auto-generated SIL Files

The SIL code for a YANG module can be generated with the **make_sil_dir_pro** script described in the next section. This script can generate 'combined' SIL files or 'split' SIL files (if the **--split** parameter is present).

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

make_sil_dir --split foo

File name	Type	Description
u_foo.c	User SIL	User-provided server instrumentation code for the 'foo' module.
u_foo.h	User SIL	User-provided external definitions for the 'foo' module. Should not edit!
y_foo.c	YumaPro SIL	YumaPro server glue code for the 'foo' module. Do not edit!
y_foo.h	YumaPro SIL	YumaPro server external definitions for the 'foo' module. Do not edit!

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

make_sil_dir foo

File name	Type	Description
foo.c	Combined YumaPro and User SIL	User-provided server instrumentation code for the 'foo' module.
foo.h	Combined YumaPro and User SIL	User-provided external definitions for the 'foo' module. Should not edit!

A SIL bundle named bundle for ***test1.yang*** *and* ***test2.yang*** would be generated using the following files:

make_sil_bundle bundle1 test1 test2

File name	Type	Description
y_bundle1.c	Bundle Wrapper	Bundle init and cleanup C code; Do not edit!
y_bundle1.h	Bundle Wrapper	Bundle init and cleanup H code; Do not edit!
u_test1.c	User SIL	User-provided server instrumentation code for the 'test1' module.
u_test1.h	User SIL	User-provided external definitions for the 'test1' module. Should not edit!
y_test1.c	YumaPro SIL	YumaPro server glue code for the 'test1' module. Do not edit!

File name	Type	Description
y_test1.h	YumaPro SIL	YumaPro server external definitions for the 'test1' module. Do not edit!
u_test2.c	User SIL	User-provided server instrumentation code for the 'test2' module.
u_test2.h	User SIL	User-provided external definitions for the 'test2' module. Should not edit!
y_test2.c	YumaPro SIL	YumaPro server glue code for the 'test2' module. Do not edit!
y_test2.h	YumaPro SIL	YumaPro server external definitions for the 'test2' module. Do not edit!

4.1.8 Basic Development Steps

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

- **Create the YANG module data model** definition, or use an existing YANG module.
  - Validate the YANG module with the **yangdump-pro** 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 **$YUMAPRO_HOME** environment variable is defined, and pointing to your YumaPro development tree.
- **Create a SIL development subtree**
  - Generate the directory structure and the Makefile with the **make_sil_dir_pro** script, installed in the **/usr/bin** directory. This step will also call **yangdump-pro** to generate the initial H and C files file the SIL.

```
Example: mydir> make_sil_dir_pro --split test
```

- **Use your text editor to fill in the device-specific instrumentation** for each object, RPC method, and notification. (In this example, edit **test/src/u_test.c**) 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/test/src> make
```

- **Install the SIL library so it is available to the netconfd-pro server.**
  - Use the 'make install' command in the SIL 'src' directory.

```
Example:  mydir/test/src> sudo make install
```

- **Run the netconfd-pro 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-pro-pro Example: load test
```

- The **netconfd-pro** server will load the specified YANG module and the SIL and make it available to all sessions.

4.2 YumaPro Source Files

This section describes the files that are contained in the **yumapro-source** package.

The server C include files are copied into **/usr/include/yumapro** when the **yumapro-sdk** package is installed. The full set of installation sources is installed in **/usr/share/yumapro/src**, if the YumaPro sources are installed.

The following table lists the files that are included within the **netconf/src** directory.
(* == These source directories are not included with all 'yumapro' source packages)

Directory	Description
agt	Server implementation modules
combo-app	Example subsystem application using DB-API and SIL-SA services
db-api	Subsystem DB-API service support
db-api-app	Example subsystem application using DB-API service
mgr *	Client implementation support
ncx	Core support for YANG, XPath, XML, JSON
netconfd-pro	YumaPro **netconfd-pro** application (main server)
platform	Platform definition files (H and Makefile support)
sil-sa	Subsystem SIL-SA service support
sil-sa-app	Example subsystem application using SIL-SA service
subsys-pro	YumaPro **netconf-subsystem-pro** application (thin client to transfer external connection to internal netconfd-pro socket)
support-save-app	Internal application to decode XML file from get-support-save
restconf	YumaPro **restconf** application (FastCGI thin client to transfer external HTTP(S) connection to internal netconfd-pro socket using RESTCONF protocol
yang-api	YumaPro **yang-api** application (FastCGI thin client to transfer external HTTP(S) connection to internal netconfd-pro socket using YANG-API protocol
yangcli-pro *	YumaPro **yangcli-pro** application (NETCONF client)
yangdiff-pro *	YumaPro **yangdiff-pro** application (YANG Semantic Compare)
yangdump-pro *	YumaPro **yangdump-pro** application (YANG Compiler)
yangdump-sdk *	YumaPro **yangdump-sdk** application (YANG Compiler with code generation support)
ycli *	Client and server YANG based CLI support
ycontrol	Subsystem YControl protocol handler
ydump *	YANG compiler support
ypgnmi *	YumaPro **ypgnmi-app** application (GO subsystem application to transfer external gNMI requests to internal **netconfd-pro** socket)
ypgrpc*	YumaPro **ypgrpc-go-app** application (GO subsystem application

	to host gRPC server and communicate with **netconfd-pro** socket)
yp-ha-app	Application to control the YP-HA mode for the local server
yp-shell *	YumaPro **yp-shell** application (server version of YANG based CLI with direct connection to main server)
ypwatcher	YumaPro **ypwatcher** application (server state monitoring program)

4.2.1 src/ncx Directory

This directory contains the code that is used to build the **libncx.so** binary shared library that is used by all YumaPro Tools programs. It handles many of the core NETCONF/YANG data structure support, including all of the YANG/YIN, XML, and XPath processing. The following table describes the purpose of each file. Refer to the actual include file (E.g., **ncx.h** in /**usr/include/yumapro)** for more details on each external function in each C source module.

To include all H files from this directory, use **libncx.h**:

```
#include “libncx.h”
```

src/ncx C Modules

C Module	Description
b64	Encoding and decoding the YANG binary data type.
blob	Encoding and decoding the SQL BLOB data type.
bobhash	Implementation of the BOB hash function.
cap	NETCONF capability definitions and support functions
cfg	NETCONF database data structures and configuration locking support.
cli	CLI parameter parsing data driven by YANG definitions.
conf	Text .conf file encoding and decoding, data driven by YANG definitions.
def_reg	Hash-driven definition registry for quick lookup support of some data structures. Contains back-pointers to the actual data.
dlq	Double linked queue support
errmgr_dict	Fast lookup data structures for -error message data
errmsg	Custom error message handling
ext	YANG extension data structure support
getbulk	Getbulk data structure support
getcb	Server Get Callback support
grp	YANG grouping data structure support
heapchk	Heap debugging support

C Module	Description
help	Automatic help text, data-driven by YANG definitions
ipaddr_typ	Canonical handler for IP address data types
json_parse	Parse a token chain as JSON Text support, as defined in RFC 4627
json_wr	JSON Output support
libncx	This is a wrapper for all NCX Library functions. Include individual H files instead to save compile time.
log	System logging support
log_syslog	SYSLOG Logging support
log_util	Logging Utilities
log_vendor_extern	Vendor Specific Logging API
log_vendor	Vendor Specific Logging support
ncx_appinfo	YumaPro Netconf Extensions (NCX) support
ncxconst	YumaPro common constants
ncx_feature	YANG feature and if-feature statement data structure support
ncx	YANG module data structure support, and some utilities
ncx_list	Support for the **ncx_list_t** data structure, used for YANG bits and **ncx:xsdlist** data types.
ncx_nmda	Support for NMDA data structures and data types
ncxmod	File Management: Controls finding and searching for YANG/YIN files, data files, and script files
ncx_num	YumaPro **ncx_num_t** data structure support. Used for processing value nodes and XPath numbers.
ncx_str	YumaPro string support.
ncxtypes	YumaPro common types
obj	YumaPro object **(obj_template_t)** data structure access
obj_dict	Unused experimantal YANG Hash support for object identification
obj_errmsg	Object specific custom error message support
obj_help	Automated object help support used with help module
op	NETCONF operations definitions and support functions
plock_cb	Partial Lock Control Block support
plock	Partial Lock support
rpc_err	NETCONF <rpc-error> data structures and support functions.
rpc	NETCONF <rpc> and <rpc-reply> data structures and support functions
runstack	Script execution stack support for yangcli-pro-pro scripts
send_buff	NETCONF send buffer function
ses	NETCONF session data structures and session access functions
ses_msg	Message buffering support for NETCONF sessions

C Module	Description
status_enum	Return Status data type
status	Error code definitions and error support functions
thd	POSIX Threads support
tk	Token chain data structures used for parsing YANG, XPath and other syntaxes.
top	Top-level XML node registration support. The <rpc> and <hello> elements are registered by the server. The <hello>, <rpc-reply> , and <notification> elements are registered by the client.
tstamp	Time and date stamp support functions
typ	YANG typedef data structures and access functions
typ_userdef	User-defined data types (callback support)
val	YumaPro value tree data structures and access functions
val_child	Support for the val_value_t data structure. **Child_hdrQ** for NCX_BT_LIST and NCX_BT_CONTAINER
val_tree	AVL Tree storage of YANG List Data support
val_unique	YANG unique-stmt support
val_util	High-level utilities for some common SIL tasks related to the value tree.
var	User variable support, used by **yangcli-pro-pro** and (TBD) XPath
version	Hardwired release train version ID
xml_msg	XML message data structures and support functions
xmlns	XML Namespace registry
xml_util	XML parse and utility functions
xml_val	High level support functions for constructing XML-ready **val_value_t** data structures
xml_wr	XML output support functions and access-control protected message generation support
xpath1	XPath 1.0 implementation
xpath1_cmp	Xpath 1.0 search support; Compare support
xpath1_fn	Xpath 1.0 search support; XPath function library
xpath1_aio	Xpath 1.0 search support fort All In One GET2 callbacks
xpath1_get2	XPath 1.0 distributed data support via GET2 callbacks
xpath1_pred	Xpath 1.0 search support; predicate support
xpath1_res	Xpath 1.0 search support; XPath result support
xpath	XPath data structures and support functions
xpath_wr	Support for generating XPath expression content within an XML instance document
xpath_yang	Special YANG XPath construct support, such as path expressions and instance identifiers
yangapi	YANG-API/RESTCONF protocols support

C Module	Description
yangconst	YANG constants
yang	YANG definitions and general support functions
yang_data	rc:yang-data extension support
yang_ext	YANG parsing and validation of the extension statement
yang_feature	Support for YANG Feature and if-feature statements
yang_grp	YANG parsing and validation of the grouping statement
yang_hash	YANG Hash support functions (unused)
yang_obj	YANG parsing and validation of the rpc, notification, and data definition statements
yang_parse	Top level YANG parse and validation support
yang_patch	YANG Patch support
yang_typ	YANG typedef and type statement support
yin	YANG to YIN mapping definitions
yinyang	YIN to YANG translation
ypgnmi	YP-GNMI protocol support

4.2.2 src/platform Directory

This directory contains platform support include files and Makefile support files. It is used by all YumaPro C modules to provide an insulating layer between YumaPro programs and the hardware platform that is used. For example the **m__getMem, m__getObj**, and **m__freeMem** macros are used instead of **malloc** and **free** functions directly.

The following table describes the files that are contained in this directory:

src/platform Files

File	Description
curversion.h	File generated during the build process to get the SVNVERSION number
platform.profile	Included by Makefiles for build support
platform.profile.cmn	Included by Makefiles for build support
platform.profile.depend	Included by Makefiles for dependency generation support
procdefs.h	Platform definitions. Contains basic data types and macros used throughout the YumaPro code. All C files include this file before any other YumaPro files.
setversion.sh	Shell script to generate the curversion.h file

4.2.3 src/agt Directory

This directory contains the NETCONF server implementation and built-in module SIL code. A static library called **libagt.a** is built and statically linked within the **netconfd-pro** program.

The following table describes the C modules contained in this directory:

src/agt C Modules

C Module	Description
agt	Server initialization and cleanup control points. Also contains the **agt_profile_t** data structure.
agt_acm_extern	External ACM API support
agt_acm	NETCONF access control implementation. Wrapper for 3 different ACM models: IETF, Yuma, and External (vendor provided)
agt_acm_ietf	IETF NACM (RFC 6536) support
agt_acm_yuma	Yuma NACM support (unused and not supported)
agt_action	Support for the YANG 1.1 action-stmt
agt_audit	Audit log support
agt_vallhome	CallHome protocol support
agt_cap	Server capabilities. Generates the server <capabilities> element content.
agt_cb	SIL callback support functions.
agt_cfg	Configuration Edit Transaction support
agt_cli	Server CLI and .conf file control functions.
agt_commit_complete	Commit Complete Callback support
agt_conf	Nested config file support
agt_connect	Handles the internal <ncx-connect> message sent from the **netconf-subsystem-pro** to the **netconfd-pro** server.
agt_crypt	Crypt-hash data type support
agt_curl	libcurl support for <url> parameter
agt_db_api	Handles the DB-API service (messages to/from subsystem)
agt_db_lock	Distributed DB-Lock feature support
agt_get2	Second generation GET support
agt_getbulk	GETBULK support
agt_hello	Handles the incoming client <hello> message and generates the server <hello> message.
agt_hook_util	NETCONF Server Set/Transaction Hook utility functions. This file contains functions to support validation of callbacks and some supplemental functions.
agt_ietf_notif	NETCONF Base notifications (RFC 6470)

C Module	Description
agt_json_parse	JSON Input support for YANG-API/RESTCONF protocols
agt_library	Setup the server in the library mode
agt_modtags	Module tags support
agt_ncchd	OpenSSH connect support for CallHome protocol
agt_ncx	NETCONF protocol operation implementation. Contains the **yuma-netconf** module SIL callback functions.
agt_ncx_load	NETCONF Server load / unload operations support
agt_ncxserver	Implements the ncxserver loop, handling the IO between the server NETCONF sessions and the **netconf-subsystem-pro** thin client program.
agt_nmda	NMDA module initialization and cleanup
agt_not	NETCONF Notifications implementation. Contains the **notifications** and **nc-notifications** modules SIL callback functions.
agt_openssl	Support for NETCONF over TYLS protocol
agt_owner	Configuration Owner support tracks client user names associated with configuration changes
agt_plock	Partial Lock (RFC 5717) support
agt_profile	Server Profile support allows
agt_restcmn	YANG-API/RESTCONF protocols common handler
agt_restconf	RESTCONF protocol handler
agt_rpc	NETCONF RPC operation handler
agt_rpcerr	NETCONF <rpc-error> generation
agt_ses	NETCONF session support and implementation of the YumaPro Session extensions. Contains the **yuma-mysession** module SIL callback functions.
agt_signal	Server signal handling support
agt_sil	Handles SIL-SA service (messages to/from subsystem) for distributed remote transactions
agt_sil_lib	Dynamic library management for SIL and SIL-SA libraries
agt_sil_profile	Handles agt_profile initialization on SIL-SA subsystems
agt_state	Standard NETCONF monitoring implementation. Contains the **ietf-netconf-monitoring** SIL callback functions.
agt_sys	Server system monitoring and notification generation. Contains the **yuma-system** module SIL callback functions.
agt_templates	Support for yumaworks-templates modules
agt_time_filter	Supports yuma-time-filter module for efficient retrieval based on last-modified timestamp of the datastore (or data node)
agt_timer	SIL periodic timer callback support functions
agt_top	Server registration and dispatch of top-level XML messages
agt_tree	Subtree filtering implementation

C Module	Description
agt_tree_get2	Support for Get2 callback data retrieval
agt_util	SIL callback utilities
agt_val	Server validation, and commit support for NETCONF database operations
agt_val_parse	Incoming <rpc> and <config> content parse and complete YANG constraint validation
agt_val_rollback	Rollback support for NETCONF database operations
agt_val_silcall	NETCONF Server database callback handler. SIL callback code
agt_val_unload	Unload module support
agt_xml	Server XML processing interface to **ncx/xml_util** functions
agt_xpath	XPath filtering implementation
agt_yangapi_edit	YANG-API/RESTCONF datastore editing support
agt_yangapi	YANG-API protocol handler
agt_yangapi_reply	YANG-API/RESTCONF response message handler
agt_yangpatch	YANG-PATCH Edit Handler for HA/RESTCONF/NETCONF
agt_ycontrol	YControl Subsystem Message handler
agt_ypcoap	CoAP (RFC 7252) Support
agt_ypgnmi	YP-GNMI protocol support
agt_ypgnmi_get	YP-GNMI protocol GET operation support
agt_ypgnmi_not	YP-GNMI protocol notification support
agt_ypgnmi_set	YP-GNMI protocol SET operation support
agt_ypgrpc	YP-GRPC protocol support
agt_ypgrpc_state	YP-GRPC monitoring implementation. Contains the **yumaworks-grpc-mon** SIL callback functions.
agt_yp_ha	High Availability module (YP-HA)
agt_yp_ha_active	High Availability module (YP-HA). HA Active Mode
agt_yp_ha_standby	High Availability module (YP-HA). HA Standby Mode
agt_ypsnmp	SNMP Protocol support
agt_ypsnmp_agentx	SNMP Protocol Agent-X support
agt_ypsnmp_not	SNMP Protocol Notification support
agt_ypsnmp_sec	SNMP Protocol Security support
agt_ypsnmp_util	SNMP Protocol utilities
ietf-netconf-nmda	NMDA protocol operations for NETCONF protocol
u_yumaworks_event_filter	Supports yumaworks-event-filter module for disabling specific notification types (User module)
u_yumaworks_templates	Supports yumaworks-templates module for template-driven configuration (User module)
y_yumaworks_event_	Supports yumaworks-event-filter module for disabling specific

C Module	Description
filter	notification types (Yumapro module)
y_yumaworks_event_ filter	Supports yumaworks-event-filter module for disabling specific notification types (Yumapro module)

4.2.4 src/mgr Directory

This module contains the NETCONF client support code. It handles all the basic NETCONF details so a simple internal API can be used by NETCONF applications such as **yangcli-pro-pro**. A static library called **libmgr.a** is built and statically linked within the **yangcli-pro** program.

To include all H files from this directory, use **libmgr.h:**

```
#include “libmgr.h”
```

The following table describes the C modules contained in this directory:

src/mgr C++ Modules

C Module	Description
c-api-devices	yp-client APIs for client device configuration
c-api-session	yp-client APIs for client session management
c-api-users	yp-client APIs for client user configuration
libmgr	Include most of the mgr libraries from 1 H file
mgr	Client initialization and cleanup control points. Also contains manager session control block data structure support functions.
mgr_callhome	Client support for NETCONF CallHome sessions
mgr_cap	Generate the client NETCONF <capabilities> element content
mgr_coap	CoAP message handler
mgr_hello	Handles the incoming server <hello> message and generates the client <hello> message.
mgr_http	HTTP message handler
mgr_io	Handles SSH server IO support for client NETCONF sessions
mgr_load	Load external file variables into the system
mgr_not	Handles incoming server <notification> messages
mgr_rpc	Generate <rpc> messages going to the NETCONF server and process incoming <rpc-reply> messages from the NETCONF server.
mgr_ses	Handles all aspects of client NETCONF sessions.
mgr_signal	Client signal handler
mgr_top	Client registration and dispatch of top-level XML messages
mgr_val_parse	Incoming <rpc-reply>, <notification>, and <config> content parse and complete YANG constraint validation.
mgr_xml	Client XML processing interface to **ncx/xml_util** functions

4.2.5 src/subsys-pro Directory

This directory contains the **netconf-subsystem-pro** program. This is a thin-client application that just transfers input and output between the SSH server and the NETCONF server. The main C source modules are called **netconf-subsystem** and **subsystem**. This is a stand-alone binary that is part of the **yumapro** package. It is installed in the **/usr/sbin/** directory.

4.2.6 src/netconfd-pro Directory

This directory contains the **netconfd-pro** program, which implements the NETCONF server. It contains one C module called **netconfd-pro**, which defines the NETCONF server 'main' function. This is a stand-alone binary that is part of the **yumapro-server** package. It is installed in the **/usr/sbin/** directory.

4.2.7 src/yangcli-pro Directory

This directory contains the **yangcli-pro** program, which is the YumaPro NETCONF client program. This is a stand-alone binary that is part of the **yumapro-client** package. It is installed in the **/usr/bin/** directory.

4.2.8 src/yangdiff-pro Directory

This directory contains the **yangdiff-pro** program, which is the YumaPro YANG module compare program. This is a stand-alone binary that is part of the **yumapro-client** package. It is installed in the **/usr/bin/** directory.

The following table describes the C modules contained in this directory:

src/yangdiff-pro

C Module	Description
yangdiff-pro	YANG module semantic compare program
yangdiff-pro_grp	Implements semantic diff for YANG grouping statement
yangdiff-pro_obj	Implements semantic diff for YANG data definition statements
yangdiff-pro_typ	Implements semantic diff for YANG typedef and type statements
yangdiff-pro_util	Utilities used by the other **yangdiff-pro** C modules

4.2.9 src/yangdump-pro Directory

This directory contains the **yangdump-pro** program, which is the YumaPro YANG compiler program. This is a stand-alone binary program. The source code is included in the YumaPro SDK Complete license. It is installed in the **/usr/bin/** directory.

4.2.10 src/yangdump-sdk Directory

This directory contains the **yangdump-sdk** program, which is the YumaPro YANG compiler program with code generation support. This is a stand-alone binary program. The source code is included in the YumaPro SDK Complete license. It is installed in the **/usr/bin/** directory.

4.2.11 src/ydump Directory

This directory contains some library files to support the **yangdump-pro** program, which is the YumaPro YANG compiler program. This is a static library linked with that program. The source code is included in the YumaPro SDK Complete license.

The following table describes the C modules contained in this directory:

src/ydump C Modules

C Module	Description
c	Implements SIL C file generation
c_util	Utilities used for SIL code generation
cyang	Handle C/H file conversion (access from ydump)
h	Implements SIL H file generation
html	Implements YANG to HTML translation
sql	Implements SQL generation for YANG module WEB Docs
xsd	Implements YANG to XSD translation
xsd_typ	Implements YANG typedef/type statement to XSD simpleType and complexType statements
xsd_util	XSD conversion utilities
xsd_yang	YANG to XSD translation utilities
yangdump_util	Utilities used by all yangdump-pro C modules
yangstats	YANG module statistics support
yangyin	Implements YANG to YIN translation
ydump	Main library entry point

4.2.12 src/ypwatcher Directory

This directory contains the **ypwatcher** program, which is the YumaPro server's state monitoring program. This is a stand-alone binary program. The source code is included in the YumaPro SDK Complete license. It is installed in the **/usr/bin/** directory.

4.2.13 src/ypgnmi Directory

This directory contains the **ypgnmi-app** program, which is the YumaPro gNMI application. This is a stand-alone binary program. It is installed in the **/usr/bin/** directory.

The following table lists the files that are included within the **netconf/src/ypgnmi** directory.

Directory	Description
gnmi	gNMI server handling, utility functions and functions to deal with the messages
gnmi_connect	gNMI server code that responsible for the gNMI client to the netconfd-pro server communication
message_handler	Auto-generated gostruct representation of the yumaworks-yp-gnmi.yang file. Used for message handling
netconfd_connect	Handler for the **netconfd-pro** connection with **ypgnmi-app**
utils	Generic utility functions
ycontrol	Utilities to handle the **netconfd-pro** YControl messages and connections

The **ypgnmi-app.go** a subsystem application that provides connectivity between the **netconfd-pro** server and gNMI clients.

4.2.14 src/ypgrpc Directory

This directory contains the **yumapro-grpc** program, which is the YumaPro gRPC application. This is a stand-alone binary program. It is installed in the **/usr/bin/** directory.

The following table lists the files that are included within the **netconf/src/ypgrpc** directory.

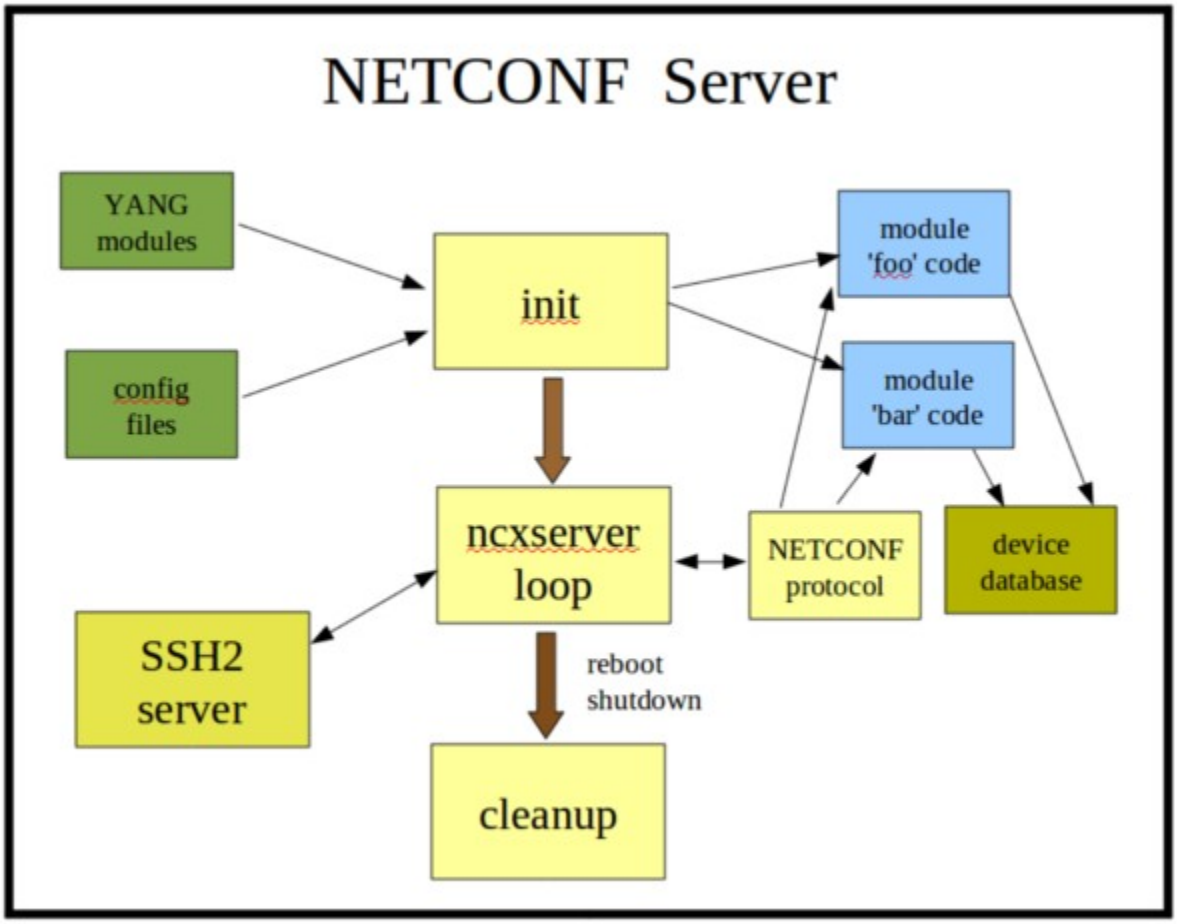
Directory	Description
cli	Handle the CLI parameters for **ypgrpc-go-app** application
credentials	Package credentials loads certificates and validates user credentials.
examples	Stub code example for Proto files (helloworld and example Protos)
log	Handle the Logging for **ypgrpc-go-app** application
message_handler	Auto-generated gostruct representation of the yumaworks-yp-grpc.yang file. Used for message handling

netconfd_connect	Handler for the **netconfd-pro** connection with **ypgrpc-go-app**
proto	Proto Files handling, parsing, search and storing
utils	Generic utility functions
ycontrol	Utilities to handle the **netconfd-pro** YControl messages and connections

The **ypgrpc-go-app.go** is a main application that provides gRPC server functionality, connectivity to the **netconfd-pro** server and stub code gRPC Services callback handling.

4.3 Server Design

This section describes the basic design used in the **netconfd-pro** server.



Initialization:

The **netconfd-pro** server will launch **ypwatcher** monitoring program, process the YANG modules, CLI parameters, config file parameters, and startup device NETCONF database, then wait for NETCONF sessions.

ncxserver Loop:

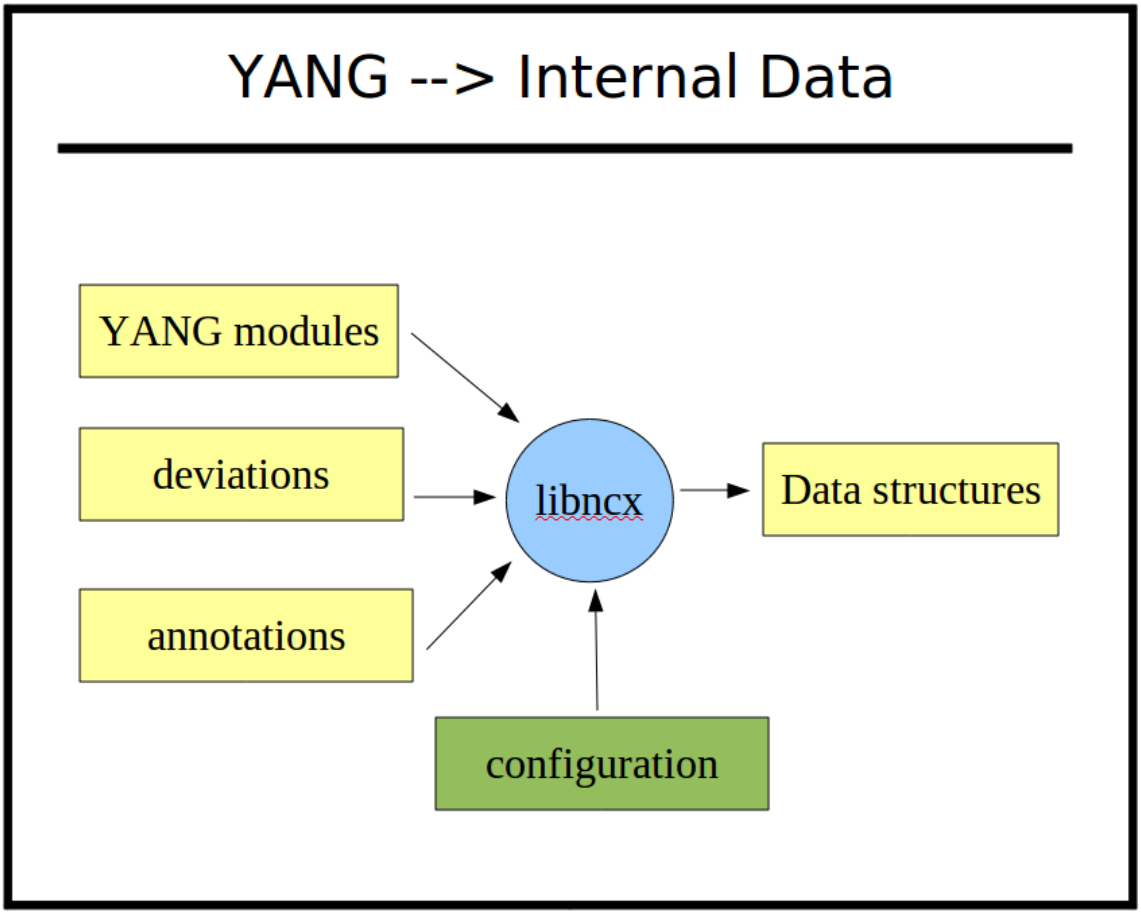
The SSH2 server will listen for incoming connections which request the 'netconf' subsystem.

When a new session request is received, the **netconf-subsystem-pro** program is called, which opens a local connection to the **netconfd-pro** server, via the ncxserver loop. NETCONF <rpc> requests are processed by the internal NETCONF stack. The module-specific callback functions (blue boxes) can be loaded into the system at build-time or run-time. This is the device instrumentation code, also called a server implementation library (SIL). For example, for **libtoaster**, this is the code that controls the toaster hardware.

Cleanup:

If the <shutdown> or <reboot> operations are invoked, then the server will cleanup. For a reboot, the init cycle is started again, instead of exiting the program.

4.3.1 YANG Native Operation



YumaPro uses YANG source modules directly to implement NETCONF protocol operations automatically within the server. The same YANG parser is used by all YumaPro programs. It is located in the 'ncx' source directory (libncx.so). There are several different parsing modes, which is set by the application.

In the 'server mode', the descriptive statements, such as 'description' and 'reference' are discarded upon input. Only the machine-readable statements are saved. All possible database validation, filtering, processing, initialization, NV-storage, and error processing is done, based on these machine readable statements.

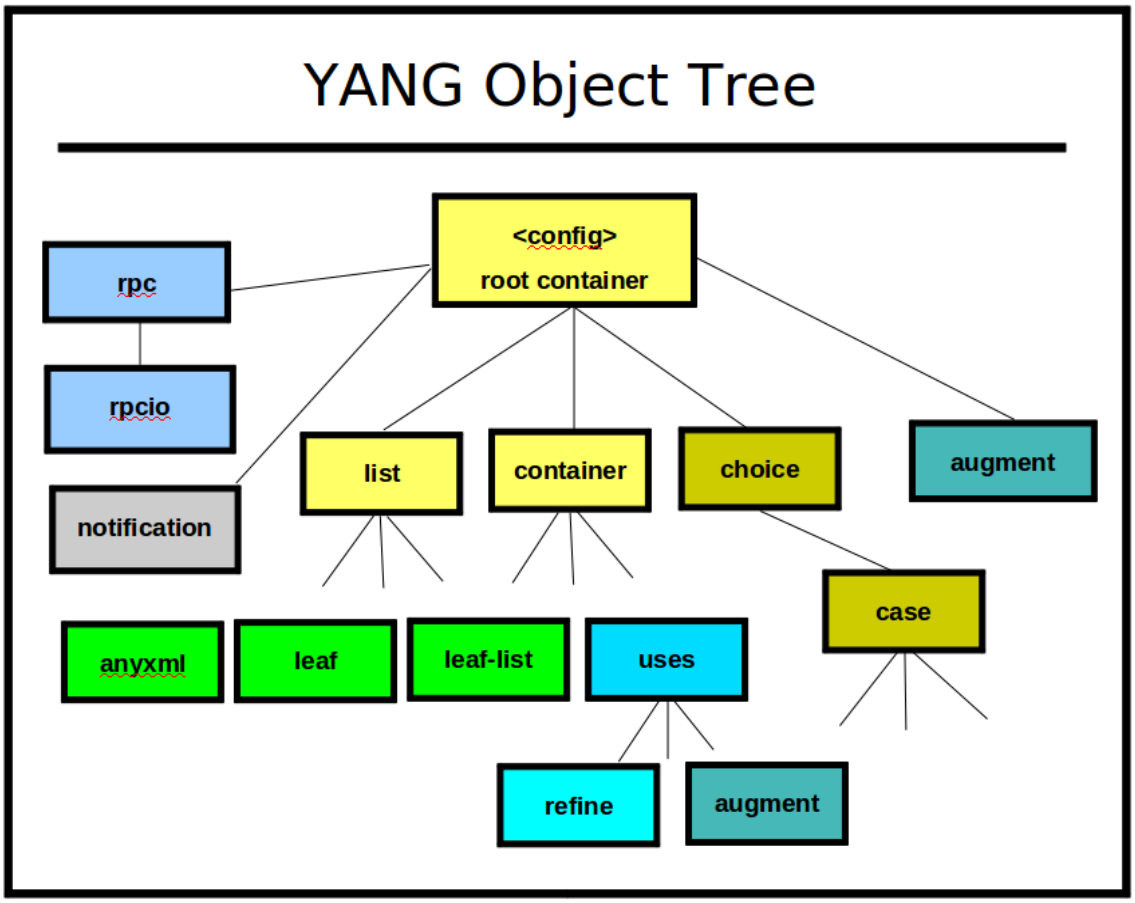
For example, in order to set the platform-specific default value for some leaf, instead of hard-coded it into the server instrumentation, the default is stored in YANG data instead. The YANG file can be altered, either directly (by editing) or indirectly (via deviation statements), and the new or altered default value specified there.

In addition, range statements, patterns, XPath expressions, and all other machine-readable statements are all processed automatically, so the YANG statements themselves are like server source code.

YANG also allows vendor and platform-specific deviations to be specified, which are like generic patches to the common YANG module for whatever purpose needed. YANG also allows annotations to be defined and added to YANG modules, which are specified with the 'extension' statement. YumaPro uses some extensions to control some automation features, but any module can define extensions, and module instrumentation code can access these annotation during server operation, to control device behavior.

There are CLI parameters that can be used to control parser behavior such as warning suppression, and protocol behavior related to the YANG content, such as XML order enforcement and NETCONF protocol operation support. These parameters are stored in the server profile, which can be customized for each platform.

4.3.2 YANG Object Tree



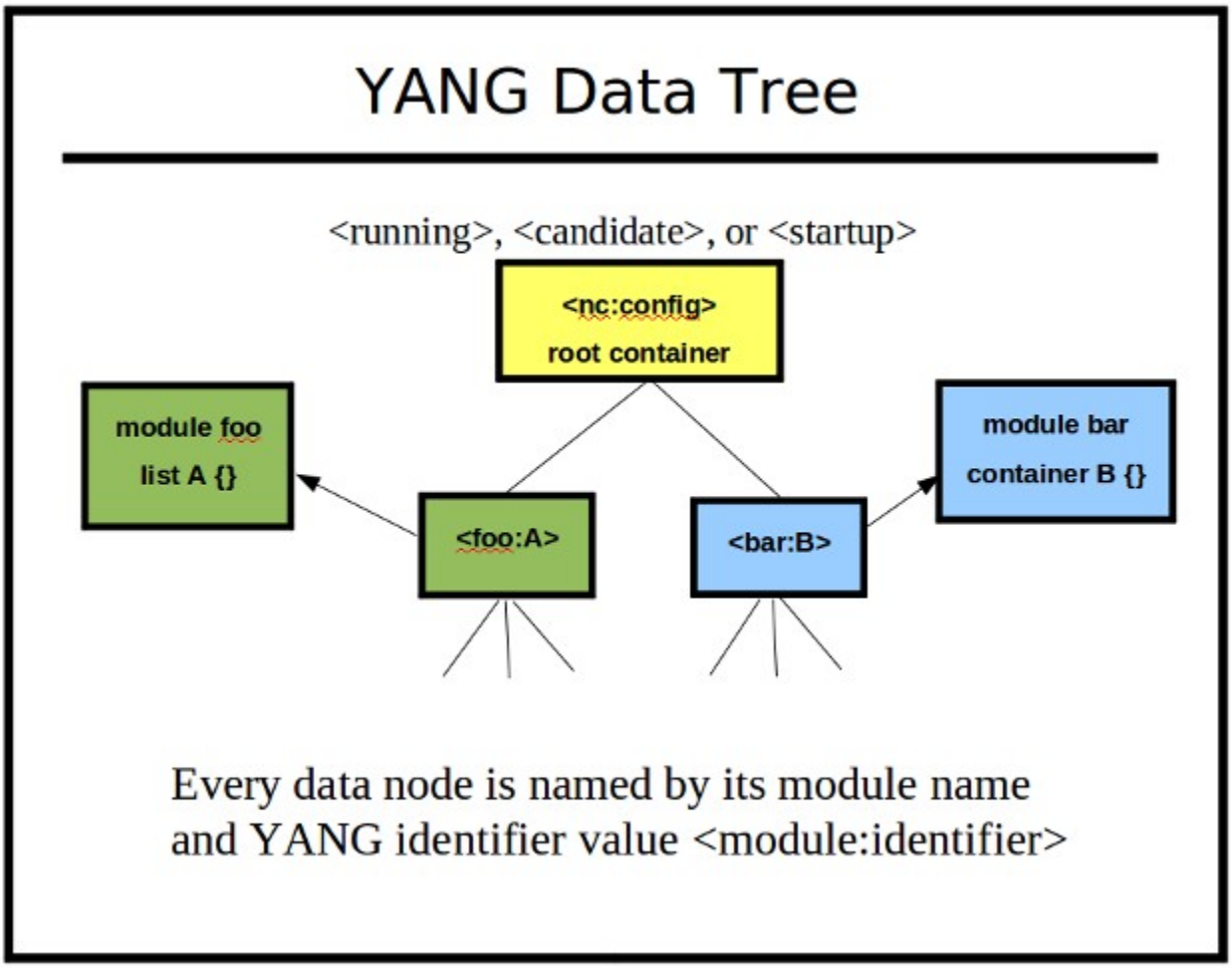
The YANG statements found in a module are converted to internal data structures.

For NETCONF and database operations, a single tree of **obj_template_t** data structures is maintained by the server. This tree represents all the NETCONF data that is supported by the server. It does not represent any actual data structure instances. It just defines the data instances that are allowed to exist on the server.

Raw YANG vs. Cooked YANG:

Some of the nodes in this tree represent the exact YANG statements that the data modeler has used, such as 'augment', 'refine', and 'uses', but these nodes are not used directly in the object tree. They exist in the object tree, but they are processed to produce a final set of YANG data statements, translated into 'cooked' nodes in the object tree. If any deviation statements are used by server implementation of a YANG data node (to change it to match the actual platform implementation of the data node), then these are also 'patched' into the cooked YANG nodes in the object tree.

4.3.3 YANG Data Tree



A YANG data tree represents the instances of 1 or more of the objects in the object tree.

Each NETCONF database is a separate data tree. A data tree is constructed for each incoming message as well. The server has automated functions to process the data tree, based on the desired NETCONF operation and the object tree node corresponding to each data node.

Every NETCONF node (including database nodes) are distinguished with XML Qualified Names (QName). The YANG module namespace is used as the XML namespace, and the YANG identifier is used as the XML local name.

Each data node contains a pointer back to its object tree schema node. The value tree is comprised of the **val_value_t** structure. Only real data is actually stored in the value tree. For example, there are no data tree nodes for choices and cases. These are conceptual layers, not real layers, within the data tree.

The NETCONF server engine accesses individual SIL callback functions through the data tree and object tree. Each data node contains a pointer to its corresponding object node.

Each data node may have several different callback functions stored in the object tree node. Usually, the actual configuration value is stored in the database, However, **virtual data nodes** are also supported. These are simply placeholder nodes within the data tree, and usually used for non-configuration nodes, such as counters. Instead of using a static value stored in the data node, a callback function is used to retrieve the instrumentation value each time it is accessed.

4.3.4 Service Layering

All of the major server functions are supported by service layers in the 'agt' or 'ncx' libraries:

- **Memory management**: macros in **platform/procdefs.h** are used instead of using direct heap functions. The macros **m__getMem** or **m__getObj** are used by YumaPro code to allocate memory. Both of these functions increment a global counter called **malloc_count**. The macro **m__free** is used to delete all malloced memory. This macro increments a global counter called **free_count**. When a YumaPro program exists, it checks if **malloc_count** equals **free_count**, and if not, generates an error message. If this occurs, the **MEMTRACE=1** parameter can be added to the make command to activate 'mtrace' debugging.
- **Queue management**: APIs in **ncx/dlq.h** are used for all double-linked queue management.
- **XML namespaces**: XML namespaces (including YANG module namespaces) are managed with functions in **ncx/xmlns.h**. An internal 'namespace ID is used internally instead of the actual URI.
- **XML parsing**: XML input processing is found in **ncx/xml_util.h** data structures and functions.
- **XML message processing**: XML message support is found in **ncx/xml_msg.h** data structures and functions.
- **XML message writing with access control**: XML message generation is controlled through API functions located in **ncx/xml_wr.h**. High level (value tree output) and low-level (individual tag output) XML output functions are provided, which hide all namespace, indentation, and other details. Access control is integrated into XML message output to enforce the configured data access policies uniformly for all RPC operations and notifications. The access control model cannot be bypassed by any dynamically loaded module server instrumentation code.
- **XPath Services**: All NETCONF XPath filtering, and all YANG XPath-based constraint validation, is handled with common data structures and API functions. The XPath 1.0 implementation is native to the server, and uses the object and value trees directly to generate XPath results for NETCONF and YANG purposes. NETCONF uses XPath differently than XSLT, and libxml2 XPath processing is memory intensive. These functions are located in **ncx/xpath.h**, **ncx/xpath1.h**, and **ncx/xpath_yang.h**. XPath filtered <get> responses are generated in **agt/agt_xpath.c**.
- **Logging service**: Encapsulates server output to a log file or to the standard output, filtered by a configurable log level. Located in **ncx/log.h**. In addition, the macro **SET_ERROR()** in **ncx/status.h** is used to report programming errors to the log.
- **Session management**: All server activity is associated with a session. The session control block and API functions are located in **ncx/ses.h**. All input, output, access control, and protocol operation support is controlled through the session control block (ses_cb_t).
- **Timer service**: A periodic timer service is available to SIL modules for performing background maintenance within the main service loop. These functions are located in **agt/agt_timer.h**.
- **Connection management**: All TCP connections to the netconfd-pro server are controlled through a main service loop, located in **agt/agt_ncxserver.c**. It is expected that the 'select' loop in this file will be replaced in embedded systems. The default netconfd-pro server actually listens for local <ncx-connect> connections on an AF_LOCAL socket. The openSSH server listens for connections on port 830 (or other configured TCP ports), and the netconf-subsystem-pro thin client acts as a conduit between the SSH server and the **netconfd-pro** server.
- **Database management**: All configuration databases use a common configuration template, defined in **ncx/cfg.h**. Locking and other generic database functions are handled in this module. The actual manipulation of the value tree is handled by API functions in **ncx/val.h**, **ncx/val_util.h, agt/agt_val_parse.h,** and **agt/agt_val.h**.
- **NETCONF operations**: All standard NETCONF RPC callback functions are located in **agt/agt_ncx.c**. All operations are completely automated, so there is no server instrumentation APIs in this file.
- **NETCONF request processing**: All <rpc> requests and replies use common data structures and APIs, found in **ncx/rpc.h** and **agt/agt_rpc.h**. Automated reply generation, automatic filter processing, and message state data is contained in the RPC message control block.

- **NETCONF error reporting:** All <rpc-error> elements use common data structures defined in **ncx/rpc_err.h** and **agt/agt_rpcerr.h.** Most errors are handled automatically, but 'description statement' semantics need to be enforced by the SIL callback functions. These functions use the API functions in **agt/agt_util.h** (such as agt_record_error) to generate data structures that will be translated to the proper <rpc-error> contents when a reply is sent.
- **YANG module library management:** All YANG modules are loaded into a common data structure (**ncx_module_t**) located in **ncx/ncxtypes.h**. The API functions in **ncx/ncxmod.h** (such as **ncxmod_load_module**) are used to locate YANG modules, parse them, and store the internal data structures in a central library. Multiple versions of the same module can be loaded at once, as required by YANG.
- **Availability monitoring service:** The **ypwatcher** is a program that provides monitoring mechanism to **netconfd-pro** server and its state. The **ypwatcher** program periodically checks the server's state and determine if the server is still running. If the server is no longer running it cleans up the state, restarts the server, and generates a syslog message.

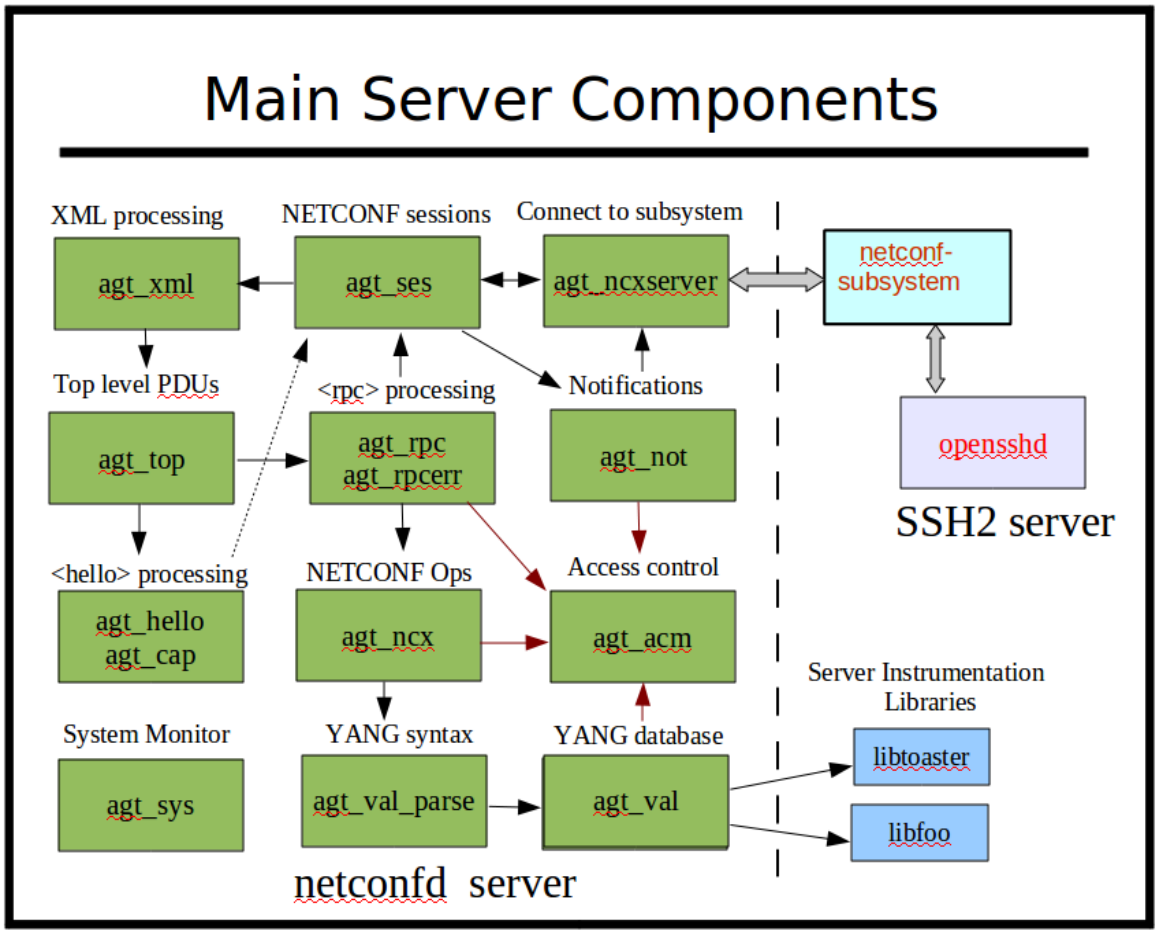
4.3.5 Session Control Block

Once a NETCONF session is started, it is assigned a session control block for the life of the session. All NETCONF and system activity in driven through this interface, so the **ncxserver** loop can be replaced in an embedded system.

Each session control block (**ses_scb_t**) controls the input and output for one session, which is associated with one SSH user name. Access control (see ietf-netconf-acm.yang) is enforced within the context of a session control block. Unauthorized return data is automatically removed from the response. Unauthorized <rpc> or database write requests are automatically rejected with an 'access-denied' error-tag.

The user preferences for each session are also stored in this data structure. They are initially derived from the server default values, but can be altered with the <set-my-session> operation and retrieved with the <get-my-session> operation.

4.3.6 Server Message Flows



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

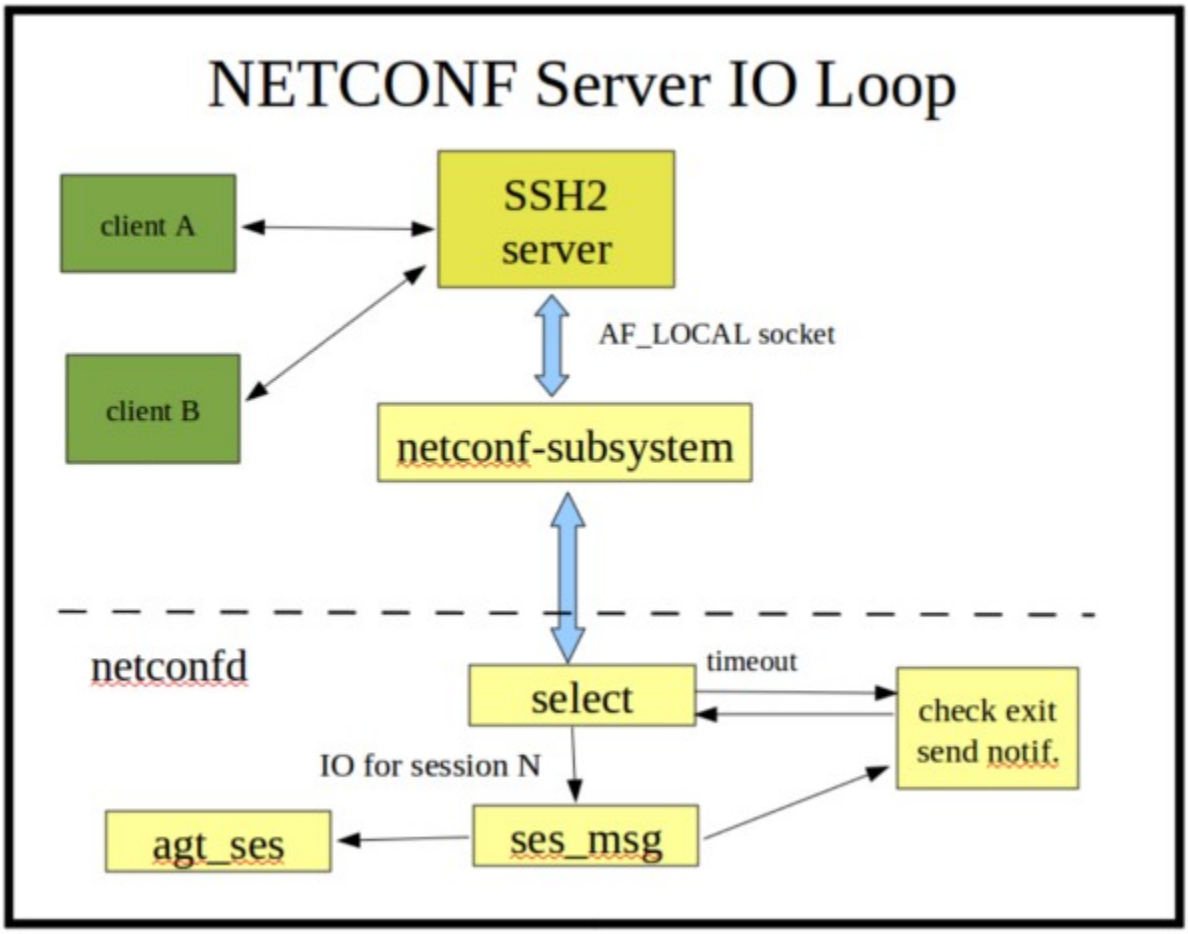
- NETCONF session management
- NETCONF/YANG database management
- NETCONF/YANG protocol operations

- 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 object management (if-feature, when)
- Memory management
- Logging management
- Timer services

All NETCONF and YANG protocol operation details are handled automatically within the **netconfd-pro** 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-pro** program. The developer fills in server callback functions to activate the networking device behavior represented by each YANG data model.

4.3.7 Main ncxserver Loop



The **ncxserver** loop does very little, and it is designed to be replaced in an embedded server that has its own SSH server:

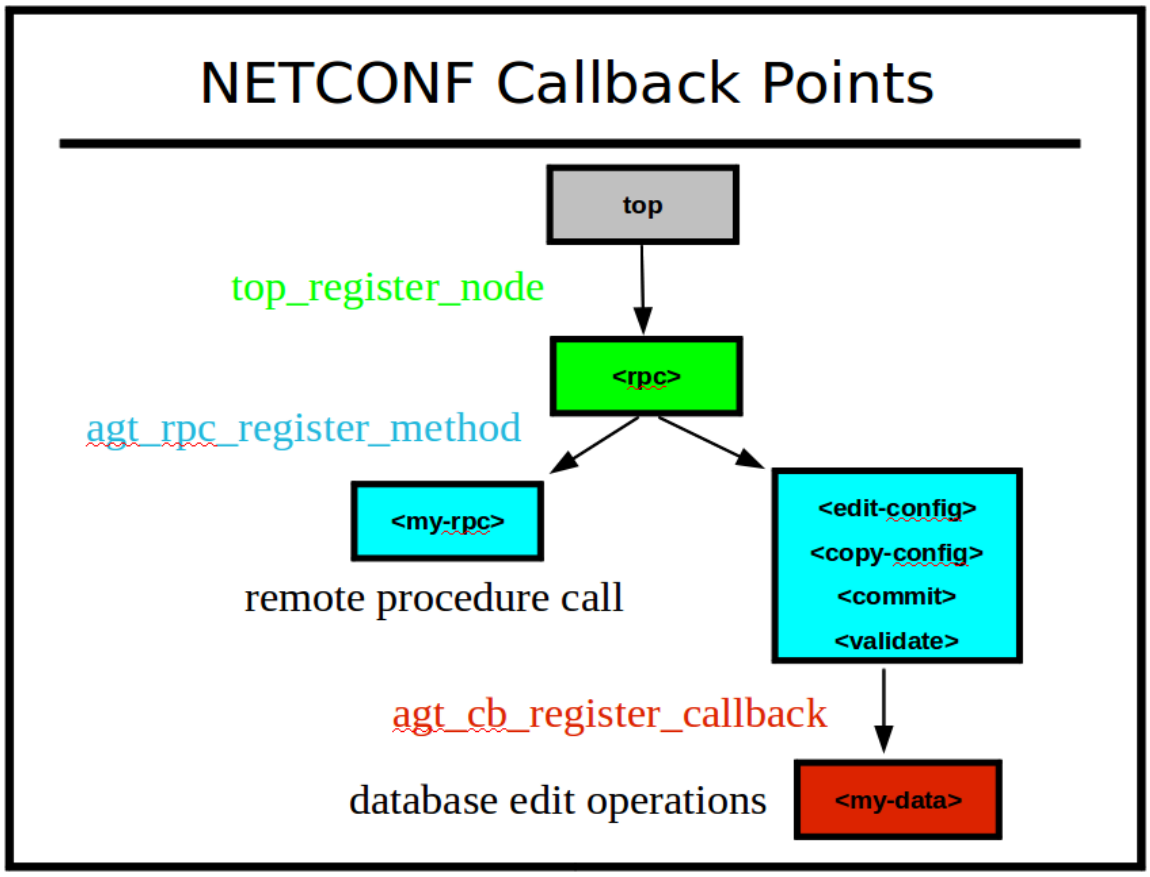
- A client request to start an SSH session results in an SSH channel being established to an instance of the **netconf-subsystem-pro** program.
- The **netconf-subsystem-pro** program will open a local socket (/**tmp/ncxserver.sock**) and send a proprietary <ncxconnect> message to the netconfd-pro server, which is listening on this local socket with a select loop (in **agt_ncxserver.c**).
- When a valid <ncxconnect> message is received by **netconfd-pro**, a new NETCONF session is created.
- After sending the <ncxconnect> message, the **netconf-subsystem-pro** program goes into 'transfer mode', and simply passes input from the SSH channel to the **netconfd-pro** server, and passes output from the **netconfd-pro** server to the SSH server.
- The **ncxserver** loop simply waits for input on the open connections, with a quick timeout. Each timeout, the server checks if a reboot, shutdown, signal, or other event occurred that needs attention.
- Notifications may also be sent during the timeout check, if any events are queued for processing. The **--max-burst** configuration parameter controls the number of notifications sent to each notification subscription, during this timeout check.

- Input <rpc> messages are buffered, and when a complete message is received (based on the NETCONF End-of-Message marker), it is processed by the server and any instrumentation module callback functions that are affected by the request.

When the agt_ncxserver_run function in **agt/agt_ncxserver.c** is replaced within an embedded system, the replacement code must handle the following tasks:

- Call **agt_ses_new_session** in **agt/agt_ses.c** when a new NETCONF session starts.
- Call **ses_accept_input** in **ncx/ses.c** with the correct session control block when NETCONF data is received.
- Call **agt_ses_process_first_ready** in **agt/agt_ses.c** after input is received. This should be called repeatedly until all serialized NETCONF messages have been processed.
- Call **agt_ses_kill_session** in **agt/agt_ses.c** when the NETCONF session is terminated.
- The following functions are used for sending NETCONF responses, if responses are buffered instead of sent directly (streamed).
  - **ses_msg_send_buffs** in **ncx/ses_msg.c** is used to output any queued send buffers.
- The following functions need to be called periodically:
  - **agt_shutdown_requested** in **agt/agt_util.c** to check if the server should terminate or reboot
  - **agt_ses_check_timeouts** in **agt/agt_ses.c** to check for idle sessions or sessions stuck waiting for a NETCONF <hello> message.
  - **agt_timer_handler** in **agt/agt_timer.c** to process server and SIL periodic callback functions.
  - **send_some_notifications** in **agt/agt_ncxserver.c** to process some outgoing notifications.

4.3.8 SIL Callback Functions



- Top Level: The top-level incoming messages are registered, not hard-wired, in the server message processing design. The **agt_ncxserve**r module accepts the <ncxconnect> message from **netconf-subsystem-pro**. The **agt_rpc** module accepts the NETCONF <rpc> message. Additional messages can be supported by the server using the **top_register_node** function.
- All RPC operations are implemented in a data-driven fashion by the server. Each NETCONF operation is handled by a separate function in **agt_ncx.c**. Any proprietary operation can be automatically supported, using the **agt_rpc_register_method** function.
  - Note: Once the YANG module is loaded into the server, all RPC operations defined in the module are available. If no SIL code is found, these will be dummy 'no-op' functions. This mode can be used to provide some server simulation capability for client applications under development.
- All database operations are performed in a structured manner, using special database access callback functions. Not all database nodes need callback functions. One callback function can be used for each 'phase', or the same function can be used for multiple phases. The **agt_cb_register_callback** function in **agt/agt_cb.c** is used by SIL code to hook into NETCONF database operations.

4.4 Server Initialization

This section describes the server initialization sequence in some detail. There are several phases in the initialization process, and various CLI parameters are defined to control the initialization behavior.

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

The platform-specific initialization code should be located in **/usr/lib/yumapro/libyp_system.so.** An example can be found in **libsystem/src/example-system.c**

The YANG module or bundle specific SIL or SIL-SA code is also initialized at startup if there are CLI parameters set to load them at boot-time.

4.4.1 Set the Server Profile

The **agt_profile_t** struct defined in agt/agt.h is used to hold the server profile parameters.

The function **init_server_profile** in **agt/agt.c** is called to set the factory defaults for the server behavior.

The function **load_extern_system_code** in **agt/agt.c** is called to load the external system library. If found, the agt_system_init_profile_fn_t callback is invoked. This allows the external system code to modify the server profile settings.

```
/* system init server profile callback
 *
 * Initialize the server profile if needed
 *
 * INPUTS:
 *  profile == server profile to change if needed
 */
typedef void (*agt_system_init_profile_fn_t)
    (agt_profile_t *profile);
```

4.4.2 Bootstrap CLI and Load Core YANG Modules

The '**ncx_init**' function is called to setup core data structures. This function also calls the **bootstrap_cli** function in **ncx/ncx.c**, which processes some key configuration parameters that need to be set right away, such as the logging parameters and the module search path.

The following common parameters are supported:

- home
- fileloc-fhs
- log-level
- log
- log-append
- log-console
- log-header
- log-mirroring
- log-stderr
- log-syslog
- log-syslog-level
- modpath
- runpath

After the search path and other key parameters are set, the server loads some core YANG modules, including netconfd-pro.yang, which has the YANG definitions for all the CLI and configuration parameters.

4.4.3 External System Init Phase I : Pre CLI

The system init1 callback is invoked with the **pre_cli** parameter set to TRUE.

This callback phase can be used to register external ACM or logging callbacks.

```
/* system init1 callback
 * init1 system call
 * this callback is invoked twice; before and after CLI processing
 * INPUTS:
 * pre_cli == TRUE if this call is before the CLI parameters
 *            have been read
 *            FALSE if this call is after the CLI parameters
 *            have been read
 * RETURNS:
 *  status; error will abort startup
 */
typedef status_t (*agt_system_init1_fn_t)
    (boolean pre_cli);
```

4.4.4 Load CLI Parameters

Any parameters entered at the command line have precedence over values set later in the configuration file(s).

The following parameters can only be entered from the command line

- The --**config** parameter specifies the configuration file to load.
- The --**no-config** parameter specifies that the default configuration file **(/etc/yumapro/netconfd-pro.conf)** should be skipped.
- The **--help** parameter causes help to be printed, and then the server will exit.
- The **--version** parameter causes the version to be printed, and then the server will exit.

The command line parameters are loaded but not activated yet.

4.4.5 Load Main Configuration File

The --**config** parameter specifies a file or default configuration file is loaded.

The command line parameters are loaded but not activated yet. Any leaf parameters already entered at the command line are skipped.

The --**confdir** parameter specifies the location of a secondary configuration directory. This parameter can be set at the command line or the main configuration file.

The command line parameters are loaded but not activated yet.

4.4.6 Load Secondary Configuration Files

The --**confdir** parameter specifies the directory to check for extra configuration files.

DO NOT SET THIS PARAMETER TO THE SAME DIRECTORY AS THE MAIN CONFIGURATION FILE.

The default value is **/etc/yumapro/netconfd-pro.d**. This directory is not created by default when the program is installed. It must be created manually with the “mkdir” command.

All files ending in “.conf” will be loaded.

The command line parameters are loaded but not activated yet.

4.4.7 Configuration Parameter Activation

After all CLI and configuration parameter files have been loaded they are applied to the **agt_profile** structure and possibly other parameters within the server.

If the --**version** or --**help** parameters are entered, the server will exit after this step.

There are a small number of parameters that can be changed at run-time. Load the **yumaworks-server** module to change these parameters at run-time.

4.4.8 External System Init Phase I : Post CLI

The system init1 callback is invoked with the **pre_cli** parameter set to FALSE.

This callback phase can be used to register most system callbacks and other preparation steps.

4.4.9 Load SIL Libraries and Invoke Phase I Callbacks

The --**module** CLI parameter specifies a module name to use for a SIL or SIL-SA library for a single module.

The --**bundle** CLI parameter specifies a bundle name to use for SIL or SIL-SA library for a set of modules.

The **--loadpath** CLI parameter specifies a set of directories to search for YANG modules, to load SIL and SIL-SA libraries from. Only modules not already loaded with **--module** or **--bundle** parameters will be loaded, if duplicate modules are found.

The init1 callbacks are invoked at this point. The SIL or SIL-SA code usually loads the appropriate YANG modules and registers callback functions.

4.4.10 External System Init Phase II : Pre-Load

The system init2 callback is invoked with the **pre_load** parameter set to TRUE.

This callback phase can be used to register an external NACM group callback and other preparation steps.

```
/* system init2 callback
 * init2 system call
 * this callback is invoked twice; before and after
 * load_running_config processing
 *
 * INPUTS:
 * pre_load == TRUE if this call is before the running config
 *             has been loaded
 *             FALSE if this call is after the running config
 *             has been loaded
 * RETURNS:
 *  status; error will abort startup
 */

typedef status_t (*agt_system_init2_fn_t)
    (boolean pre_load);
```

4.4.11 Load <running> Configuration

The --**startup** parameter specifies the XML file containing the YANG configuration data to load into the <running> datastore.

The --**no-startup** parameter specifies that this step will be skipped.

The --**no-nvstore** parameter specifies that this step will be skipped.

The --**factory-startup** parameter will cause the server to use the --**factory-startup-file** parameter value, or the default **factory-startup-cfg.xml**, or an empty configuration if none of these files are found.

If none of these parameters is entered, the default location is checked.

If an external handler is configured by registering an **agt_nvload_fn** callback, this this function will be called to get the startup XML file, instead of the **--startup** of default locations.

If --**fileloc-fhs**=true, then the default file is **/var/lib/netconfd-pro/startup-cfg.xml.** If false, then the default file is **$HOME/.yumapro/startup-cfg.xml**

If the default file **startup-cfg.xml** is not found, the the file **factory-startup-cfg.xml** will be checked, unless the --**startup-factory-file** parameter is set.

If the --**startup-prune-ok** parameter is set to TRUE, then unknown data nodes will be pruned from the <running> configuration. Otherwise unknown data nodes will cause the server to terminate with an error.

If the **--startup-error** parameter is set to continue, then errors in the parsing of the configuration data do not cause the server to terminate, if possible. If **--startup-error** is set to 'fallback' then the server will attempt to reboot with the factory default configuration.

If the **--running-error** parameter is set to continue, then errors in the root-check validation of the configuration data do not cause the server to terminate, if possible.

4.4.12 Invoke SIL Phase II Callbacks

The phase II callbacks for SIL and SIL-SA libraries are invoked.

This callback usually checks the <running> datastore and activates and configuration found.

Operational data is initialized in this phase as well.

4.4.13 External System Init Phase II : Post-Load

The system init2 callback is invoked with the **pre_load** parameter set to FALSE.

This callback phase can be used to load data into the server.

4.4.14 Initialize SIL-SA Subsystems

At this point, the server is able to accept control sessions from subsystems such as DB-API and SIL-SA. The init1 and init2 callbacks will be invoked for SIL-SA libraries. The SIL-SA must register callbacks in the init1 callback so the main server knows to include the subsystem in edit transactions. Phase II initialization for SIL-SA libraries is limited to activating the startup configuration data.

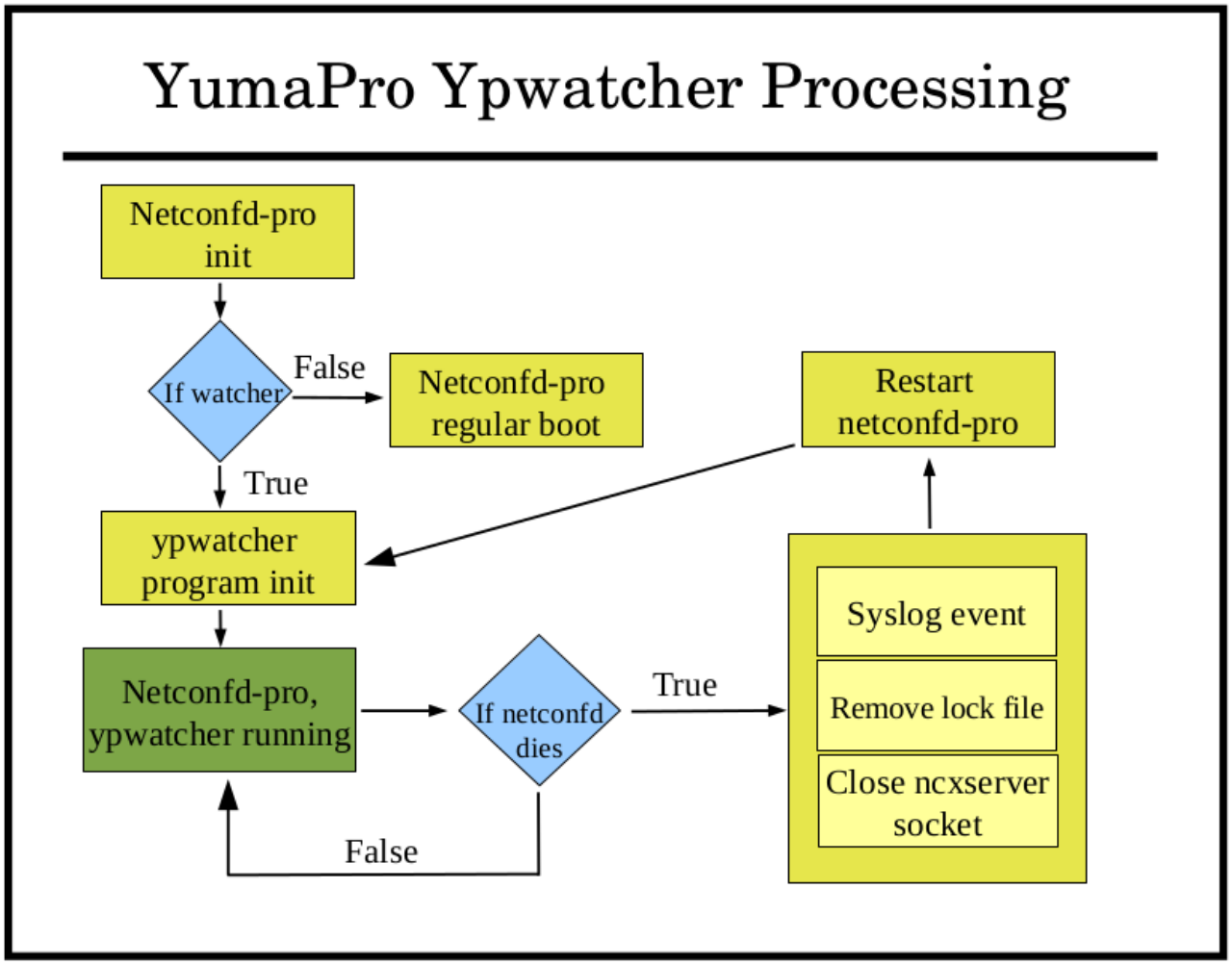
4.4.15 Server Ready

At this point the server is ready to accept management sessions for the configured northbound protocols.

4.5 Server Operation

This section briefly describes the server internal behavior for some basic NETCONF operations.

4.5.1 Ypwatcher processing



After modules definition and configuration parameters are loaded successfully, the **ypwatcher** program is called to start the monitoring process.

After **agt_cli_process_input** function process **netconfd-pro** input parameters. In order to process **ypwatcher** specific parameters this function checks for the existence of the watcher parameters in the input.

The **ypwatcher** program will be called by the **agt_init1** function by default unless **--no-watcher** parameter will be specified or the program is already running.

```
netconfd-pro --no-watcher
```

The **ypwatcher** program is running continuously and attempting to restart the server any time it exits unexpectedly.

Unexpected exit can be interpreted as a server's shut down process due to severe error, such as Segmentation fault, core dump, bus error, and invalid memory reference. **Ypwatcher** will restart the server only if any of these termination actions causing the server to shut down.

4.5.2 Loading Modules and SIL Code

YANG modules and their associated device instrumentation can be loaded dynamically with the **--module** configuration parameter. Some examples are shown below:

```
module=foo
module=bar
module=baz@2009-01-05
module=~/mymodules/myfoo.yang
```

- The **ncxmod_find_sil_file** function in **ncx/ncxmod.c** is used to find the library code associated with the each module name. The following search sequence is followed:
  - Check the **$YUMAPRO_HOME/target/lib** directory
  - Check each directory in the **$YUMAPRO_RUNPATH** environment variable or **--runpath** configuration variable.
  - Check the **/usr/lib/yumapro** directory
- If the module parameter contains any sub-directories or a file extension, then it is treated as a file, and the module search path will not be used. Instead the absolute or relative file specification will be used.
- If the first term starts with an environment variable or the tilde (~) character, and will be expanded first
- If the 'at sign' (@) followed by a revision date is present, then that exact revision will be loaded.
- If no file extension or directories are specified, then the module search path is checked for YANG and YIN files that match. The first match will be used, which may not be the newest, depending on the actual search path sequence.
- The **$YUMAPRO_MODPATH** environment variable or **--modpath** configuration parameter can be used to configure one or more directory sub-trees to be searched.
- The **$YUMAPRO_HOME** environment variable or **--yumapro-home** configuration parameter can be used to specify the YumaPro project tree to use if nothing is found in the currect directory or the module search path.
- The **$YUMAPRO_INSTALL** environment variable or default YumaPro install location (/usr/share/yumapro/modules) will be used as a last resort to find a YANG or YIN file.

The server processes **--module** parameters by first checking if a dynamic library can be found which has an 'soname' that matches the module name. If so, then the SIL phase 1 initialization function is called, and that function is expected to call the **ncxmod_load_module** function.

If no SIL file can be found for the module, then the server will load the YANG module anyway, and support database operations for the module, for provisioning purposes. Any RPC operations defined in the module will also be accepted (depending on access control settings), but the action will not actually be performed. Only the input parameters will be checked, and <or> or some <rpc-error> returned.

4.5.3 Core Module Initialization

The **agt_init2** function in **agt/agt.c** is called after the configuration parameters have been collected.

- Initialize the core server code modules
- Static device-specific modules can be added to the agt_init2 function after the core modules have been initialized
- Any 'module' parameters found in the CLI or server configuration file are processed.
- The **agt_cap_set_modules** function in **agt/agt_cap.c** is called to set the initial module capabilities for the **ietf-netconf-monitoring** module

4.5.4 Startup Configuration Processing

After the static and dynamic server modules are loaded, the **--startup** (or **--no-startup**) parameter is processed by **agt_init2** in **agt/agt.c**:

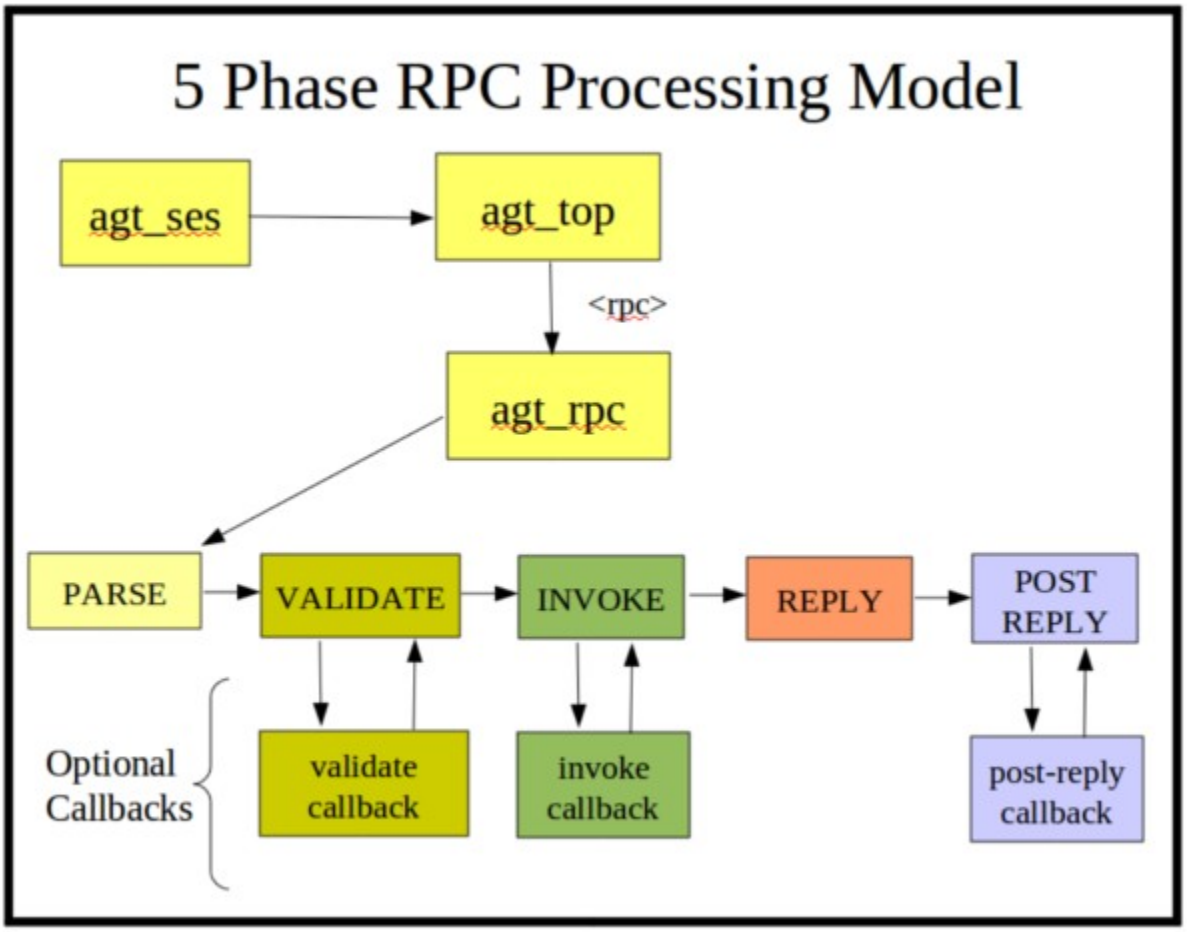
- If the **--startup** parameter is used and includes any sub-directories, it is treated as a file and must be found, as specified.
- Otherwise, the **$YUMAPRO_DATAPATH** environment variable or **--datapath** configuration parameter can be used to determine where to find the startup configuration file.
- If neither the **--startup** or **--no-startup** configuration parameter is present, then the data search path will be used to find the default **startup-cfg.xml**
- The **$YUMAPRO_HOME** environment variable or **--yumapro-home** configuration parameter is checked if no file is found in the data search path. The **$YUMAPRO_HOME/data** directory is checked if this parameter is set.
- The **$YUMAPRO_INSTALL** environment variable is checked next, if the startup configuration is still not found.
- The default location if none is found is the **$HOME/.yumapro/startup_cfg.xml** file.

It is a fatal error if a startup config is specified and it cannot be found.

As the startup configuration is loaded, any SIL callbacks that have been registered will be invoked for the association data present in the startup configuration file.. The edit operation will be OP_EDITOP_LOAD during this callback.

After the startup configuration is loaded into the running configuration database, all the stage 2 initialization routines are called. These are needed for modules which add read-only data nodes to the tree containing the running configuration. SIL modules may also use their 'init2' function to create factory default configuration nodes (which can be saved for the next reboot).

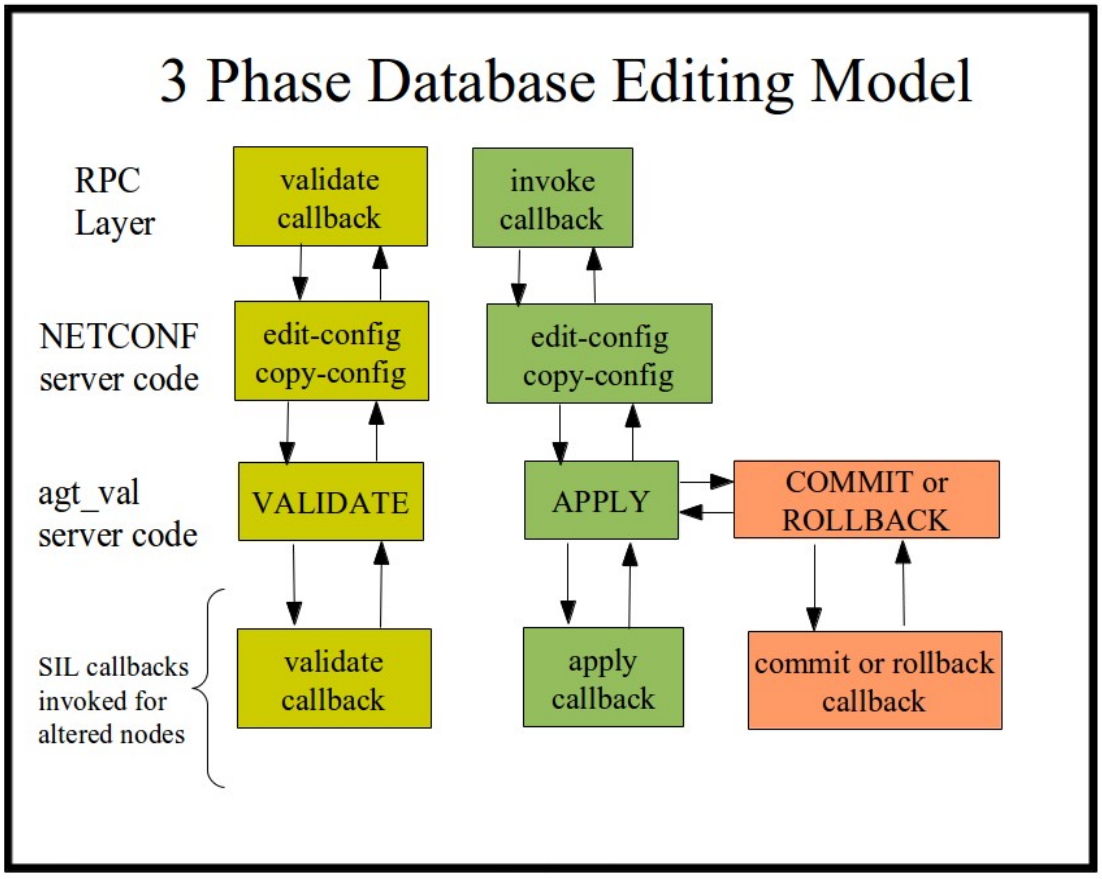
4.5.5 Process an Incoming <rpc> Request



- **PARSE Phase:** The incoming buffer is converted to a stream of XML nodes, using the **xmlTextReader** functions from **libxml2**. The agt_val_parse function is used to convert the stream of XML nodes to a **val_value_t** structure, representing the incoming request according to the YANG definition for the RPC operation. An **rpc_msg_t** structure is also built for the request.
- **VALIDATE Phase:** If a message is parsed correctly, then the incoming message is validated according to the YANG machine-readable constraints. Any description statement constraints need to be checked with a callback function. The **agt_rpc_register_method** function in **agt/agt_rpc.c** is used to register callback functions.
- **INVOKE Phase:** If the message is validated correctly, then the invoke callback is executed. This is usually the only required callback function. Without it, the RPC operation has no affect. This callback will set fields in the **rpc_msg_t** header that will allow the server to construct or stream the <rpc-reply> message back to the client.
- **REPLY Phase:** Unless some catastrophic error occurs, the server will generate an <rpc-reply> response. If any <rpc-error> elements are needed, they are generated first. If there is any response data to send, that is generated or streamed (via callback function provided earlier) at this time. Any unauthorized data (according to to the **ietf-netconf-acm.yang** module configuration) will be silently dropped from the message payload. If there were no errors and no data to send, then an <ok> response is generated.

- **POST_REPLY Phase:** After the response has been sent, a rarely-used callback function can be invoked to cleanup any memory allocation or other data-model related tasks. For example, if the **rpc_user1** or **rpc_user2** pointers in the message header contain allocated memory then they need to be freed at this time.

4.5.6 Edit the Database



- **Validate Phase:** The server will determine the edit operation and the actual nodes in the target database (candidate or running) that will be affected by the operation. All of the machine-readable YANG statements which apply to the affected node(s) are tested against the incoming PDU and the target database. If there are no errors, the server will search for a SIL validate callback function for the affected node(s). If the SIL code has registered a database callback function for the node or its local ancestors, it will be invoked. This SIL callback function usually checks additional constraints that are contained in the YANG description statements for the database objects.
- **Apply Phase:** If the validate phase completes without errors, then the requested changes are applied to the target database.
  - Note: This phase is used for the internal data tree manipulation and validation only. It is not used to alter device behavior. Resources may need to be reserved during the SIL apply callback, but the database changes are not activated at this time.
- **Commit or Rollback Phase:** If the validate and apply phases complete without errors, then then the server will search for SIL commit callback functions for the affected node(s) in the target database. This SIL callback phase is

used to apply the changes to the device and/or network. It is only called when a commit procedure is attempted. This can be due to a <commit> operation, or an <edit-config> or <copy-config> operation on the running database.

- Note: If there are errors during the commit phase, then the backup configuration will be applied, and the server will search for a SIL callback to invoke with a 'rollback operation'. The same procedure is used for confirmed commit operations which timeout or canceled by the client.

4.5.7 Save the Database

The following bullets describe how the server saves configuration changes to non-volatile storage:

- If the **--with-startup=true** parameter is used, then the server will support the :startup capability. In this case, the <copy-config> command needs to be used to cause the running configuration to be saved.
- If the **--with-startup=false** parameter is used, then the server will not support the :startup capability. In this case, the database will be saved each time the running configuration is changed.
- The <copy-config> or <commit> operations will cause the startup configuration file to be saved, even if nothing has changed. This allows an operator to replace a corrupted or missing startup configuration file at any time.
- The database is saved with the **agt_ncx_cfg_save** function in **agt/agt_ncx.c.**
- The **with-defaults** 'explicit' mode is used during the save operation to filter the database contents.
  - Any values that have been set by the client will be saved in NV-storage.
  - Any value set by the server to a YANG default value will not be saved in the database.
  - If the server create a node that does not have a YANG default value (E.g., containers, lists, keys), then this node will be saved in NV storage.
- If the **--startup=filespec** parameter is used, then the server will save the database by overwriting that file. The file will be renamed to backup-cfg.xml first.
- If the **--no-startup** parameter is used, or no startup file is specified and no default is found, then the server will create a file called 'startup-cfg.xml', in the following manner:
  - If the **$YUMAPRO_HOME** variable is set, the configuration will be saved in **$YUMAPRO_HOME/data/startup-cfg.xml**.
  - Otherwise, the configuration will be saved in **$HOME/.yumapro/startup-cfg.xml.**
- The database is saved as an XML instance document, using the <config> element in the NETCONF 'base' namespace as the root element. Each top-level YANG module supported by the server, which contains some explicit configuration data, will be saved as a child node of the <nc:config> element. There is no particular order to the top-level data model elements.

4.6 Built-in Server Modules

There are several YANG modules which are implemented within the server, and not loaded at run-time like a dynamic SIL module. Some of them are IETF standard modules and some are YumaPro extension modules.

4.6.1 iana-crypt-hash.yang

This module contains the crypt-hash typedef. This data type is supported internally by the server. Passwords configured using this data type are stored as hash values. The plain-text passwords are never stored if this data type is used.

4.6.2 ietf-datastores.yang

This module contains the datastore-ref typedef from RFC 6244. This data type is supported internally by the server.

4.6.3 ietf-inet-types.yang

This module contains the standard YANG Internet address types, and is defined in RFC 6021. These types are available for commonly used management object types. A YANG module author should check this module first, before creating any new data types with the YANG typedef statement.

There are no accessible objects in this module, so there are no SIL callback functions. The YANG data-types are supported within the YumaPro engine core modules, such as **ncx/val.c** and **ncx/xml_wr.c.**

4.6.4 ietf-origin.yang

This module contains the origin typedef from RFC 6244. This data type is supported internally by the server.

4.6.5 ietf-netconf.yang

The standard NETCONF module is used to access the NETCONF operations, and is defined in RFC 6536.

The module yuma-netconf.yang is actually used instead, even though this module is advertised to clients.

The **agt/agt_ncx.c** contains the SIL callback functions for this module.

4.6.6 ietf-netconf-acm.yang

The standard NETCONF Access Control module is used to control which YANG content each group of users can access from the server. The base module is supported, and is defined in RFC 6536.

This YANG module is used by default, is the 'ietf' ACM mode is selected, which is the default ACM mode in YumaPro.

The **agt/agt_acm_ietf.c** contains the SIL callback functions for this module.

4.6.7 ietf-netconf-nmda.yang

The standard NETCONF operations for NMDA module is used to implement the <get-data> operation from RFC 8527.

The **agt/ietf-netconf-nmda.c** contains the SIL callback functions for this module.

4.6.8 ietf-netconf-monitoring.yang

The standard NETCONF Monitoring module is used to examine the capabilities, current state, and statistics related to the NETCONF server. The entire module is supported, and is defined in RFC 6022.

This module is also used to retrieve the actual YANG or YIN files (or URLs for them) that the server is using. Clients can use the <get-schema> RPC operation to retrieve the YANG or YIN files listed in the **/netconf-state/schemas** subtree. A client will normally check the <hello> message from the server for module capabilities, and use its own local copy of a server YANG module, if it can. If not, then the <get-schema> function can be used to retrieve the YANG module.

The **agt/agt_state.c** contains the SIL callback functions for this module.

4.6.9 ietf-netconf-notifications.yang

The standard NETCONF Base Notifications module is used to report common NETCONF server events in the standard NETCONF notification stream. The entire module is supported, and is defined in RFC 6470.

The **agt/agt_sys.c** contains the SIL callback functions for this module.

4.6.10 ietf-netconf-partial-lock.yang

The standard NETCONF Partial Lock module is used to lock subtrees within the <running> datastore, using the <partial-lock> and <partial-unlock> operations defined in RFC 5717.

The **agt/agt_plock.c** contains the SIL callback functions for this module.

4.6.11 ietf-netconf-with-defaults.yang

The standard <with-defaults> extension to some NETCONF operations is defined in this module. The entire module is supported, and defined in RFC 6243.

This parameter is added to the <get>, <get-config>, and <copy-config> operations to let the client control how 'default leafs' are returned by the server. The YumaPro server can be configured to use any of the default handling styles (report-all, trim, or explicit). The filtering of default nodes is handled automatically by the server support functions in **agt/agt_util.c**, and the XML write functions in **ncx/xml_wr.c.**

4.6.12 ietf-restconf.yang

The standard YANG extension “yang-data” is defined in this module, from RFC 8040. It is fully supported with the server, and has the same effect as the **ncx:abstract** extension. The module **ncx/yang_data.c** contains the code supporting this YANG extension.

4.6.13 ietf-restconf-monitoring.yang

This module contains the /restconf-state subtree, from RFC 8040. It is fully supported and provides information about the RESTCONF capabilities and notification streams. The separate library libietf-restmonitoring contains the SIL callbacks for this module.

4.6.14 ietf-yang-library.yang

This module contains the YANG Module Library (/**modules-state** subtree), from RFC 7895. It is fully supported and provides information about the YANG modules loaded on the server. The file **libietf-yang-library/src/ietf-yang-library.c** contains the SIL callback functions for this module.

4.6.15 ietf-yang-patch.yang

This module contains the YANG Patch Media Type, defined in RFC 8072. It is fully supported and provides the template for complex RESTCONF protocol edit requests. The DB-API protocol uses the yang-patch grouping as well. The file **ncx/yang_patch.c** contains basic support for this module. The file **atg/agt_yangpatch.c** provides RESTCONF support for this module.

4.6.16 ietf-yang-types.yang

This module contains the standard YANG general user data types. These types are available for commonly used derived types. A YANG module author should check this module first, before creating any new data types with the YANG typedef statement.

There are no accessible objects in this module, so there are no SIL callback functions. The YANG data-types are supported within the YumaPro engine core modules, such as **ncx/val.c** and **ncx/xml_wr.c.**

4.6.17 nc-notifications.yang

This module is defined in RFC 5277, the NETCONF Notifications specification. It contains the <replayComplete> and <notificationComplete> notification event definitions.

The file **agt/agt_not.c** contains the SIL support code for this module.

4.6.18 netconfd-pro.yang

This module defines the CLI parameters used by netconfd-pro.

There are no NETCONF writable parameters in this module. The **sysNetconfServerCLI** object in the **yuma-system.yang** module contains a copy of the parameters that were passed to the server.

There are no SIL callbacks for this module. The CLI handling code is in **agt/agt_cli.c.**

The CLI parameters defined in this module can also be used in the **netconfd-pro.conf** configuration file loaded by the server at boot-time. The configuration file handling code is in **ncx/conf.c.**

4.6.19 notifications.yang

This module is defined in RFC 5277, the NETCONF Notifications specification. All of this RFC is supported in the server. This module contains the <create-subscription> RPC operation. The notification replay feature is controlled with the **--eventlog-size** configuration parameter. The <create-subscription> operation is fully supported, including XPath and subtree filters. The **ietf-netconf-acm** module can be used to control what notification events a user is allowed to receive. The <create-subscription> filter allows the client to select which notification events it wants to receive.

The file **agt/agt_not.c** contains the SIL callback functions for this modules.

4.6.20 yang-data-ext.yang

This module contains the “augment-yang-data” YANG extension. It is fully supported and used to augment data structures defined with the “yang-data” extension. The file **ncx/yang_data.c** contains the internal callback functions for this module.

4.6.21 yuma-app-common.yang

This module contains some common groupings of CLI parameters supported by some or all YumaPro programs. Each program with CLI parameters defines its own module of CLI parameters (using the ncx:cli extension). The program name is used for the YANG module name as well (E.g., yangdump-pro.yang or netconfd-pro.yang).

The SIL callback functions for the common groupings in this module are found in **ncx/val_util.c**, such as the **val_set_feature_parms** function.

4.6.22 yuma-ncx.yang

This module provides the YANG language extension statements that are used by YumaPro programs to automate certain parts of the NETCONF protocol, document generation, code generation, etc.

There are no SIL callback functions for this module. There are support functions within the src/ncx directory that include the obj_set_ncx_flags function in ncx/obj.c

4.6.23 yuma-mysession.yang

This module provides the YumaPro proprietary <get-my-session> and <set-my-session> RPC operations. These are used by the client to set some session output preferences, such as the desired line length, indentation amount, and defaults handling behavior. This is not a built-in module. It has to be loaded with the **--module** parameter.

The SIL callbacks for this module are located in the separate library **libyuma-mysession**.

4.6.24 yuma-netconf.yang

The NETCONF protocol operations, message structures, and error information are all data-driven, based on the YANG statements in the **yuma-netconf.yang** module. The ietf-netconf.yang module is not used at this time because it does not contain the complete set of YANG statements needed. The yuma-netconf.yang version is a super-set of the IETF version. Only one YANG module can be associated with an XML namespace in YumaPro. In a future version, the extra data structures will be moved to an annotation module.

The file **agt/agt_ncx.c** contains the SIL callback functions for this module.

This module is not advertised in the server capabilities. It is only used internally within the server.

4.6.25 yuma-system.yang

This module contains the YumaPro **/system** data structure, providing basic server information, unix 'uname' data, and all the YumaPro proprietary notification event definitions.

The file **agt/agt_sys.c** contains the SIL callback functions for this module.

4.6.26 yuma-time-filter.yang

This module contains the YumaPro **last-modified** leaf, which extends the standard **/netconf-state/datastores/datastore** structure in **ietf-netconf-monitoring.yang,** with the database last-modified timestamp. The standard <get> and <get-config> operations are augmented with the **if-modified-since** leaf, to allow all-or-none filtering of the configuration, based on its modification timestamp.

The file **agt/agt_sys.c** contains the SIL callback functions for this module.

4.6.27 yuma-types.yang

This module provides some common data types that are used by other YumaPro YANG modules.

There are no SIL callback functions for this module.

4.6.28 yumaworks-agt-profile.yang

This module contains a representation of major agt_profile data structure fields. This is an internal module used between the server and SIL-SA subsystems. This module is not advertised to clients.

4.6.29 yumaworks-app-common.yang

This module contains some common groupings of CLI parameters supported by some or all YumaPro programs. Each program with CLI parameters defines its own module of CLI parameters (using the ncx:cli extension). The program name is used for the YANG module name as well (E.g., **yangdump-pro.yang** or **netconfd-pro.yang**).

The definitions in this module are separated from the **yuma-app-common.yang** definitions because these CLI parameters and typedefs are only used in YumaPro, and not shared with Yuma programs.

There are no SIL callbacks in this module.

4.6.30 yumaworks-attrs.yang

This module contains abstract YANG definitions used to support XML attributes used with various protocols.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.31 yumaworks-config-change.yang

This module contains extra data that can be added to <netconf-config-change> notifications. This extra data represents the old and new values associated with the edit. The **--with-yumaworks-config-change** parameter must be set to true for this module to be loaded by the server.

4.6.32 yumaworks-db-api.yang

This module contains YControl protocol messages for the DB-API service.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.33 yumaworks-event-filter.yang

This module contains configuration parameters to suppress generation of specified notifications.

The **--with-yumaworks-event-filter** parameter must be set to false to disable this module.

4.6.34 yumaworks-extensions.yang

This module provides some of the YANG language extension statements that are used by YumaPro programs to automate certain parts of the NETCONF protocol, document generation, code generation, etc.

There are no SIL callback functions for this module. There are support functions within the **src/ncx** directory that include the **obj_set_ncx_flags** function in **ncx/obj.c**

The definitions in this module are separated from the yuma-ncx.yang definitions because these extensions are only used in YumaPro, and not shared with Yuma programs.

4.6.35 yumaworks-getbulk.yang

This module contains the <get-bulk> operation for NETCONF and RESTCONF.

The **--with-yumaworks-getbulk** parameter must be set to false to disable this module.

4.6.36 yumaworks-ids.yang

This module contains additional transport identities for NETCONF sessions in the ietf-netconf-monitoring module.

The **--with-yumaworks-ids** parameter must be set to false to disable this module.

4.6.37 yumaworks-internal.yang

This module contains internal RPC operation definitions used by the server.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.38 yumaworks-mgr-common.yang

This module contains internal groupings for CLI parameters used by client applications.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.39 yumaworks-opsmgr.yang

This module contains YP-Controller operations support.

This is an experimental module used for yp-controller only. This module is not advertised to clients.

This is not a built-in module. It is implemented as a stand-alone SIL that must be explicitly loaded into yp-controller.

4.6.40 yumaworks-restconf-commit.yang

This module contains confirmed commit support for the RESTCONF protocol.

This is an YumaWorks extension to the RESTCONF protocol. This module is not advertised to clients.

4.6.41 yumaworks-restconf.yang

This module contains meta-data for supporting the URIs accessible for the RESTCONF protocol.

This is an YumaWorks extension to the RESTCONF protocol. This module is not advertised to clients.

4.6.42 yumaworks-server.yang

This module contains configuration parameters that mirror the server CLI and .conf file parameters. It allows some parameters to be changed at run-time and the rest to be changed for the next reboot.

This is not a built-in module. It is implemented as a stand-alone SIL that must be explicitly loaded into the server.

4.6.43 yumaworks-sesmgr.yang

This module contains YP-Controller session support.

This is an experimental module used for **yp-controller** only. This module is not advertised to clients.

This is not a built-in module. It is implemented as a stand-alone SIL that must be explicitly loaded into **yp-controller**.

4.6.44 yumaworks-sil-sa.yang

This module contains YControl protocol messages for the SIL-SA service.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.45 yumaworks-support-save.yang

This module contains the <get-support-save> RPC operation.

The **--with-support-save** parameter must be set to false to disable this module.

4.6.46 yumaworks-system.yang

This module contains several NETCONF protocol operation augmentations and system operations like <backup> and <restore>, and load/unload operations for modules and bundles.

The **--with-yumaworks-system** parameter must be set to false to disable this module.

4.6.47 yumaworks-templates.yang

This module contains configuration template support.

The **--with-yumaworks-templates** parameter must be set to false to disable this module.

4.6.48 yumaworks-term-msg.yang

This module contains the ‘term-msg’ notification. This is used with yp-shell to support SIL-generated terminal messages to be printed directly on the console.

The **--with-term-msg** parameter must be set to false to disable this module.

4.6.49 yumaworks-test.yang

This module contains data structure definitions used by yangcli-pro to store test-suite information.

This is an internal module used for **yangcli-pro** implementation only. This module is not used by the server.

4.6.50 yumaworks-types.yang

This module contains common typedefs used by multiple programs to represent CLI parameters.

It is used by some server modules and may be advertised as needed.

4.6.51 yumaworks-yang-api.yang

This module contains meta-data for supporting the URIs accessible for the YANG-API protocol.

This is an obsolete module. It is not used by the server.

4.6.52 yumaworks-yangmap.yang

This module contains data structure definitions used by **yangcli-pro** to store YANG data model mappings to support the **--yangmap** feature in **yp-shell.** This is an internal module used for **yp-shell** implementation only. This module is not used by the server.

4.6.53 yumaworks-ycontrol.yang

This module contains YControl protocol messages for the the base protocol.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.54 yumaworks-yp-gnmi.yang

This module contains gNMI message definitions from the gNMI specification.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.55 yumaworks-yp-grpc.yang

This module contains gRPC message definitions.

This is an internal module used for implementation only. This module is not advertised to clients.

4.6.56 yumaworks-grpc-mon.yang

This module contains gRPC Monitoring Information that can be retrieved from the **netconfd-pro** server.

This module contains the **/grpc-state** subtree. It provides information about the gRPC server capabilities and streams. The separate library **agt_ypgrpc_state** contains the SIL callbacks for this module.

4.6.57 yumaworks-yp-ha.yang

This module contains YControl protocol messages for the YP-HA service.

This is an internal module used for implementation only. This module is not advertised to clients.

4.7 Optional Server Modules

There are YANG modules which are implemented outside the server in order to improve modularity and provide a choice of modules that can supplement the server with a new feature at start up or at run time. Some of them are NETCONF standard modules and some are YumaPro extension modules.

4.7.1 yuma-arp.yang

This module contains a collection of YANG definitions for configuring and monitoring ARP.

The file **libyuma-arp/src/yuma-arp.c** contains the SIL callback functions for this module.

Use **WITH_YUMA_ARP=1** flag in order to build the libyuma-arp directory and install the SIL code for the yuma-arp YANG module. Refer to README for build instructions.

4.7.2 yuma-proc.yang

This module provides some Unix **/proc** file-system data, in nested XML format. This module will not load if the files **/proc/meminfo** and **/proc/cpuinfo** are not found.

The file **libyuma-proc/src/yuma-proc.c** contains the SIL callback functions for this module.

Use **WITH_YUMA_PROC=1** flag in order to build the libyuma-proc directory and install the SIL code for the yuma-proc YANG module. Refer to README for build instructions.

4.7.3 yuma-interfaces.yang

This module contains the YumaPro interfaces table, which is just a skeleton configuration list, plus some basic interface counters. This module is intended to provide an example for embedded developers to replace this module with their own interfaces table. The YumaPro table uses information in some files found in Unix systems which support the **/proc/net/dev** system file.

The file **libyuma-interfaces/src/yuma-interfaces.c** contains the SIL callback functions for this module.

Use **WITH_YUMA_INTERFACES=1** in order to build the libyuma-interfaces directory and install the SIL code for the yuma-interfaces YANG module. Refer to README for build instructions.

4.7.4 ietf-restconf-monitoring.yang

This module contains monitoring information for the RESTCONF protocol. Contains a list of protocol capability URIs and description of stream content.

The file **libietf-restmonitoring/src/ietf-restconf-monitoring.c** contains the SIL callback functions for this module.

Use **WITH_RESTCONF=1** flag in order to build the libietf-restmonitoring directory and install the SIL code for the ietf-restconf-moniroting YANG module. Refer to README for build instructions.

5 YANG Objects and Data Nodes

This section describes the basic design of the YANG object tree and the corresponding data tree that represents instances of various object nodes that the client or the server can create.

5.1 Object Definition Tree

The object tree is a tree representation of all the YANG module rpc, data definition, and notification statements. It starts with a 'root' container. This is defined with a YANG container statement which has an **ncx:root** extension statement within it. The <config> parameter within the <edit-config> operation is an example of an object node which is treated as a root container. Each configuration database maintained by the server (E.g., <candidate> and <running>) has a root container value node as its top-level object.

A root container does not have any child nodes defined in it within the YANG file. However, the YumaPro tools will treat this special container as if any top-level YANG data node is allowed to be a child node of the 'root' container type.

5.1.1 Object Node Types

There are 14 different YANG object node types, and a discriminated union of sub-data structures contains fields common to each sub-type. Object templates are defined in ncx/obj.h.

YANG Object Types

object type	description
OBJ_TYP_ANYXML	This object represents a YANG **anyxml** data node.
OBJ_TYP_CONTAINER	This object represents a YANG presence or non-presence **container.**
OBJ_TYP_CONTAINER + ncx:root	If the ncx:root extension is present within a container definition, then the object represents a NETCONF database **root**. No child nodes
OBJ_TYP_LEAF	This object represents a YANG **leaf** data node.
OBJ_TYP_LEAF_LIST	This object represents a YANG **leaf-list** data node.
OBJ_TYP_LIST	This object represents a YANG **list** data node.
OBJ_TYP_CHOICE	This object represents a YANG **choice** schema node. The only children allowed are case objects. This object does not have instances in the data tree.
OBJ_TYP_CASE	This object represents a YANG **case** schema node. This object does not have instances in the data tree.
OBJ_TYP_USES	This object represents a YANG **uses** schema node. The contents of the grouping it represents will be expanded into object tree. It is saved in the object tree even during operation, in order for the expanded objects to share common data. This object does not have instances in the data tree.
OBJ_TYP_REFINE	This object represents a YANG **refine** statement. It is used to alter the grouping contents during the expansion of a uses

	statement. This object is only allowed to be a child of a uses statement. It does not have instances in the data tree.
OBJ_TYP_AUGMENT	This object represents a YANG **augment** statement. It is used to add additional objects to an existing data structure. This object is only allowed to be a child of a uses statement or a child of a 'root' container. It does not have instances in the data tree, however any children of the augment node will generate object nodes that have instances in the data tree.
OBJ_TYP_RPC	This object represents a YANG **rpc** statement. It is used to define new <rpc> operations. This object will only appear as a child of a 'root' container. It does not have instances in the data tree. Only 'rpcio' nodes are allowed to be children of an RPC node.
OBJ_TYP_RPCIO	This object represents a YANG **input** or **output** statement. It is used to define new <rpc> operations. This object will only appear as a child of an RPC node. It does not have instances in the data tree.
OBJ_TYP_NOTIF	This object represents a YANG **notification** statement. It is used to define new <notification> event types. This object will only appear as a child of a 'root' container. It does not have instances in the data tree.

5.1.2 Object Node Template (obj_template_t)

The following typedef is used to represent an object tree node:

```
/* One YANG data-def-stmt */
typedef struct obj_template_t_ {
    dlq_hdr_t      qhdr;
    obj_type_t     objtype;
    uint32         flags;              /* see OBJ_FL_* definitions */
    ncx_error_t    tkerr;
    grp_template_t *grp;          /* non-NULL == in a grp.datadefQ */

    /* 4 back pointers */
    struct obj_template_t_ *parent;
    struct obj_template_t_ *usesobj;
    struct obj_template_t_ *augobj;
    struct xpath_pcb_t_    *when;           /* optional when clause */
    dlq_hdr_t               metadataQ;       /* Q of obj_metadata_t */
    dlq_hdr_t               appinfoQ;         /* Q of ncx_appinfo_t */
    dlq_hdr_t               iffeatureQ;     /* Q of ncx_iffeature_t */

    /* cbset is agt_rpc_cbset_t for RPC or agt_cb_fnset_t for OBJ */
    void                   *cbset;

    /* object namespace ID assigned at runtime
     * this can be changed over and over as a
     * uses statement is expanded.  The final
     * expansion into a real object will leave
     * the correct value in place
     */
    xmlns_id_t             nsid;

    union def_ {
       obj_container_t   *container;
       obj_leaf_t        *leaf;
       obj_leaflist_t    *leaflist;
       obj_list_t        *list;
       obj_choice_t      *choic;
       obj_case_t        *cas;
       obj_uses_t        *uses;
       obj_refine_t      *refine;
       obj_augment_t     *augment;
       obj_rpc_t         *rpc;
       obj_rpcio_t       *rpcio;
       obj_notif_t       *notif;
    } def;

} obj_template_t;
```

The following table highlights the fields within the obj_template_t data structure:

obj_template_t Fields

Field	Description
qhdr	Queue header to allow the object template to be stored in a child queue
objtype	enumeration to identify which variant of the 'def' union is present
flags	Internal state and properties
tkerr	Error message information
grp	back-pointer to parent group if this is a top-level data node within a grouping
parent	Parent node if any
usesobj	Back pointer to uses object if this is a top-level data node within an expanded grouping
augobj	Back pointer to augment object if this is a top-level data node within an expanded augment
when	XPath structure for YANG when statement
metadataQ	Queue of obj_template_t for any XML attributes (ncx:metadata) defined for this object node
appinfoQ	Queue of ncx_appinfo_t for any YANG extensions found defined within the object, that were not collected within a deeper appinfoQ (E.g., within a type statement)
iffeatureQ	Queue of ncx_iffeature_t for any if-feature statements found within this object node
cbset	Set of server callback functions for this object node.
nsid	Object node namespace ID assigned by xmlns.c
def	Union of object type specific nodes containing the rest of the YANG statements. Note that the server discards all descriptive statements such as description, reference, contact,.

5.1.3 obj_template_t Access Functions

The file **ncx/obj.h** contains many API functions so that object properties do not have to be accessed directly. The following table highlights the most commonly used functions. Refer to the H file for a complete definition of each API function.

obj_template_t Access Functions

Function	Description
obj_find_template	Find a top-level object template within a module.
obj_find_child	Find the specified child node within a complex object template . Skips over any nodes without names (augment, uses, etc.). Uses the module name that defines the **obj_template_t** to find a child.
obj_find_child_fast	Find the specified child node within a complex object template . Skips over any nodes without names (augment, uses, etc.). Uses the module namespace ID that to find a child.
obj_first_child	Get the first child node within a complex object template. Skips over any nodes without names.
obj_next_child	Get the next child node after the current specified child. Skips over any nodes without names.
obj_first_terminal_child	Get the first TERMINAL child object within a complex object template. Skips over any nodes without names.
obj_next_terminal_child	Get the next TERMINAL child object within a complex object template. Skips over any nodes without names.
obj_first_terminal_child_nokey	Get the first TERMINAL child object within a complex object template. Skips over any nodes without names and KEY leafs.
obj_next_terminal_child_nokey	Get the next TERMINAL child object within a complex object template. Skips over any nodes without names and KEY leafs.
obj_first_child_augok	Get the first child object within a complex object template. Skips over “refine” and “uses” nodes.
obj_next_child_augok	Get the next child object if the specified object has any children; return augment, not just obj_has_name. Skips over “refine” and “uses” nodes.
obj_first_child_deep	Get the first child node within a complex object template . Skips over any nodes without names, and also any choice and case nodes.
obj_next_child_deep	Get the next child object after the current specified child. Skips over any nodes without names, and also any choice and case nodes.
obj_previous_child	Get the previous child object if the specified object has any children. Skips over any nodes without names.
obj_last_child	Get the LAST child object within a complex object template. Skips over any nodes without names.

Function	Description
obj_find_case	Find the specified case object child object within the specific complex object node.
obj_find_type	Check if a **typ_template_t** in the obj typedefQ hierarchy.
obj_find_grouping	Check if a **grp_template_t** in the obj typedefQ hierarchy.
obj_find_key	Find a specific key component by key leaf identifier name.
obj_first_key	Get the first **obj_key_t** structure for the specified list object type.
obj_next_key	Get the next **obj_key_t** structure for the specified list object type.
obj_gen_object_id	Allocate and generate the YANG object ID for an object node.
obj_gen_object_id_prefix	Allocate and generate the YANG object ID for an object node. Uses the prefix in every node.
obj_gen_object_id_oid	Allocate and generate the YANG object ID for an object node. Uses the JSON/YANG module-name for prefix convention . Only print module name when namespace changes.
obj_gen_object_id_xpath	Allocate and generate the YANG object ID for an object node. Remove all conceptual choice and case nodes so the resulting string will represent the structure of the value tree for XPath searching.
obj_get_name	Get the object name string.
obj_set_name	Set the name field for this object.
obj_has_name	Return TRUE if the object has a name field.
obj_has_text_content	Return TRUE if the object has text content.
obj_get_status	Get the YANG status for the object.
obj_get_description	Get the YANG description statement for an object. Note that the server will always return a NULL pointer.
obj_get_alt_description	Get the alternate YANG description field for this obj ect. Check if any 'info', then 'help' appinfo nodes present.
obj_get_reference	Get the YANG reference statement for an object. Note that the server will always return a NULL pointer.
obj_get_config_flag	Get the YANG config statement value for an object.
obj_get_max_access	Get the YANG NCX max-access enumeration for an object. Return the explicit value or the inherited value.
obj_get_typestr	Get the name string for the type of an object.
obj_get_default	Get the YANG default value for an object.
obj_get_next_default	Get the next YANG default value for the specified object. Only leaf-list object type is supported.
obj_get_default_case	Get the name of the default case for a choice object.
obj_npcon_has_defaults	Check if the specified NP container has defaults within it. Must be a

Function	Description
	config object.
obj_get_level	Get the nest level for the specified object . Top-level is '1' . Does not count groupings as a level.
obj_has_typedefs	Check if the object has any nested typedefs in it . This must only be called if the object is defined in a grouping.
obj_get_typdef	Get the internal type definition for the leaf or leaf-list object.
obj_get_basetype	Get the internal base type enumeration for an object.
obj_get_mod_prefix	Get the module prefix for an object.
obj_get_mod_xmlprefix	Get the module prefix (XML format) for this object.
obj_get_mod_name	Get the module name containing an object.
obj_get_mod_version	Get the module revision date for the module containing an object.
obj_in_submodule	Check if the object is defined in a submodule.
obj_get_nsid	Get the internal XML namespace ID for an object.
obj_get_min_elements	Get the YANG min-elements value for a list or leaf-list object.
obj_get_max_elements	Get the YANG max-elements value for a list or leaf-list object.
obj_get_units	Get the YANG units field for a leaf or leaf-list object.
obj_get_parent	Get the parent object node for an object.
obj_get_real_parent	Get the parent object node for an object. Skips over choice and case.
obj_get_presence_string	Get the YANG presence statement for a container object.
obj_get_child_count	Get the number of child nodes for a complex object.
obj_get_fraction_digits	Get the YANG fraction-digits statement for a decimal64 leaf or leaf-list object.
obj_is_anyxml	Return TRUE if the object is YANG anyxml type.
obj_is_anydata	Return TRUE if the object is YANG anydata type.
obj_is_leaf	Return TRUE if the object is YANG leaf type.
obj_is_crypt_hash	Check if the object is a leaf of type crypt-hash. Return TRUE if the object is YANG crypt-hash leaf type.
obj_is_list	Return TRUE if the object is YANG list type.
obj_is_key	Return TRUE if the object is YANG key type.
obj_is_leafy	Return TRUE if the object is YANG leaf or leaf-list type.
obj_get_leaf_list_defset	Get the defset flag for a leaf-list. Return TRUE if leaf-list with original defaults.
obj_find_defval	Find a default for leaf-list. Return TRUE if found.
obj_is_container	Return TRUE if the object is YANG container type.
obj_is_choice	Return TRUE if the object is YANG choice type.
obj_is_case	Return TRUE if the object is YANG case type.
obj_is_uses	Return TRUE if the object is YANG uses type.
obj_is_terminal	Return TRUE if the object is YANG leaf, leaf-list, anyxml, or anydata

Function	Description
	type.
obj_is_mandatory	Return TRUE if the object is YANG mandatory statement.
obj_is_mandatory_when	Return TRUE if the object is YANG mandatory, but first check if any when statements are FALSE first.
obj_has_when_stmts	Check if any when statements apply to this object . Does not check if they are true, just any when statements present. Return TRUE if object has any when statement associated with it.
obj_is_cloned	Return TRUE if the object is expanded from a grouping or augment statement.
obj_is_augclone	Return TRUE if the object is expanded from an augment statement.
obj_is_augment	Return TRUE if the object is YANG augment statement.
obj_is_external_augment	Check if an object is an external augment. Return TRUE if an augment to another module.
obj_is_external_data_augment	Check if an object is an external augment of a data node. Return TRUE if an augment to data node in another module.
obj_is_refine	Return TRUE if the object is YANG refine statement.
obj_is_data	Check if the object is defined within data or within a notification or RPC instead.
obj_is_data_db	Return TRUE if the object is defined within a YANG database definition.
obj_is_data_node	Return TRUE if the object is a real node type.
obj_is_rpc	Return TRUE if the object is a YANG RPC type.
obj_rpc_has_input	Check if the RPC object has any real input children. Return TRUE if there are any input children.
obj_rpc_has_output	Check if the RPC object has any real output children. Return TRUE if there are any output children.
obj_is_rpcio	Return TRUE if the object is a YANG RPCIO type.
obj_is_action	Return TRUE if the object is a YANG action type.
obj_is_notif	Return TRUE if the object is a YANG notification type.
obj_is_empty	Return TRUE if object is empty of subclauses.
obj_in_rpc	Return TRUE if the object is defined within an RPC statement.
obj_in_notif	Return TRUE if the object is defined within a notification statement.
obj_is_xsdlist	Return TRUE if the object is marked as an XSD list.
obj_is_hidden	Return TRUE if object contains the ncx:hidden extension.
obj_is_root	Return TRUE if object contains the ncx:root extension.
obj_is_password	Return TRUE if object contains the ncx:password extension.
obj_is_cli	Return TRUE if object contains the ncx:cli extension.
obj_is_abstract	Return TRUE if object contains the ncx:abstract extension.
obj_is_xpath_string	Return TRUE if the object is a leaf or leaf-list containing an XPath string.

Function	Description
obj_is_schema_instance_string	Return TRUE if the object is a leaf or leaf-list containing a schema instance identifier string.
obj_is_secure	Return TRUE if object contains the nacm:secure extension.
obj_is_very_secure	Return TRUE if object contains the nacm:very-secure extension.
obj_is_sil_delete_children_first	Check if object is marked as ncx:sil-delete-children-first. Return TRUE if object contains ncx:sil-delete-children-first extension.
obj_is_block_user_create	Check if object is marked as ncx:user-write with create access disabled. Return TRUE if object is marked to block user create access.
obj_is_block_user_update	Check if object is marked as ncx:user-write with update access disabled. Return TRUE if object is marked to block user update access.
obj_is_block_user_delete	Check if object is marked as ncx:user-write with delete access disabled. Return TRUE if object is marked to block user delete access.
obj_is_system_ordered	Return TRUE if the list or leaf-list object is system ordered; FALSE if it is user ordered.
obj_is_np_container	Return TRUE if the object is a YANG non presence container.
obj_is_p_container	Return TRUE if the object is a YANG presence container.
obj_is_enabled	Return TRUE if the object is enabled; FALSE if any if-feature, when-stmt, or deviation-stmt has removed the object from the system.
obj_has_iffeature	Return TRUE if object has any if-feature statements.
obj_is_single_instance	Return TRUE if object is a single instance object.
obj_sort_children	Rearrange any child nodes in YANG schema order
obj_is_top	Check if the object is top-level object within the YANG module that defines it. Return TRUE if obj is a top-level object.
obj_is_datapath	Check if object is marked as a ywx:datapath object. Return TRUE if object is marked as datapath.
obj_has_children	Check if there are any accessible nodes within the object. Return TRUE if there are any accessible children.
obj_has_ro_children	Check if there are any accessible read-only child nodes within the object. Return TRUE if there are any accessible read-only children.

Function	Description
obj_has_ro_descendants	Check if there are any accessible read-only descendant nodes within the object. Return TRUE if there are any accessible read-only descendant.
obj_has_rw_children	Check if there are any accessible read-write child nodes within the object. Do not count list keys in this check. Return TRUE if there are any accessible read-write children.
obj_enabled_child_count	Get the count of the number of enabled child nodes for the object template. Returns number of enabled child nodes.
obj_get_keystr	Get the key string for this list object. Returns pointer to key string or NULL if none or not a list.
obj_is_supported	Check an RPC node to check if it is supported or not . It could be disabled at run-time without removing it. Return TRUE if object is supported.
obj_is_dup_local	Check if this object is one that has a duplicate sibling with the same local-name and different module namespace. Return if flagged as duplicate local-name.
obj_is_exclusive_rpc	Return TRUE if exclusive-rpc set; cannot be called by multiple sessions at once.
obj_parent_same_module	Check if the object parent object is the same. Return TRUE if parent from the same module or not sure. Return FALSE if parent not the same module and parent is not root.
obj_set_sil_priority	Set the SIL priority field.
obj_get_sil_priority	Get the SIL priority field.
obj_has_config_defaults	Check if the object itself or child nodes can have a default. Return TRUE if can have a default.

5.2 Data Tree

A YumaPro data tree is a representation of some subset of all possible object instances that a server is maintaining within a configuration database or other structure.

Each data tree starts with a 'root' container, and any child nodes represent top-level YANG module data nodes that exist within the server.

Each configuration database maintains its own copy (and version) of the data tree. There is only one object tree, however, and all data trees use the same object tree for reference.

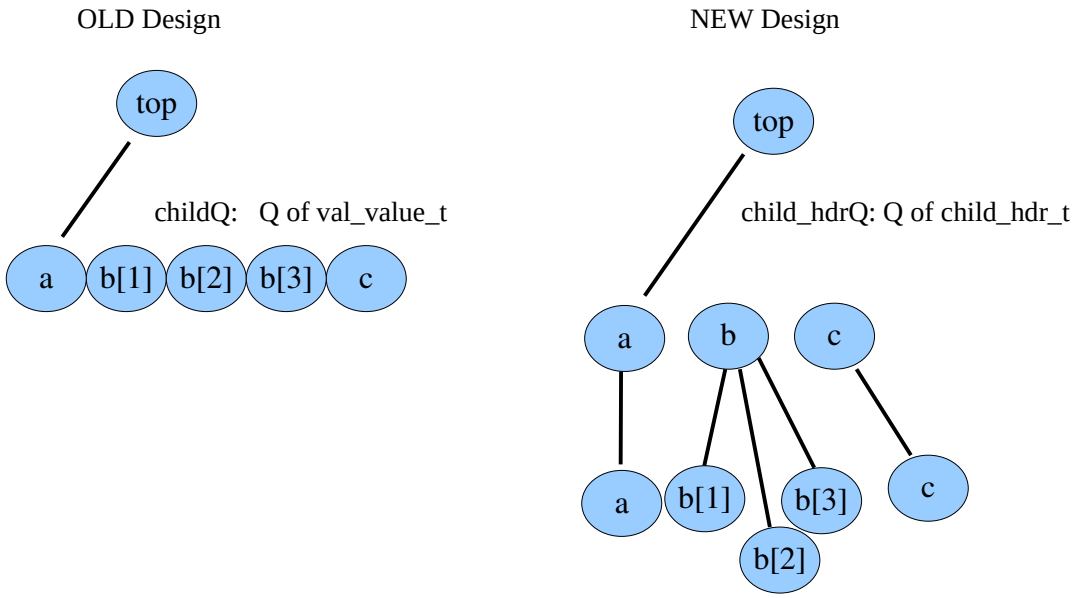
Not all object types have a corresponding node within a data tree. Only 'real' data nodes are present. Object nodes that are used as meta-data to organize the object tree (E.g., choice, augment) are not present. The following table lists the object types and whether each one is found in a data tree.

Object Types in the Data Tree

Object Type	Found In Data Tree?
OBJ_TYP_ANYXML	Yes
OBJ_TYP_CONTAINER	Yes
OBJ_TYP_CONTAINER (ncx:root)	Yes
OBJ_TYP_LEAF	Yes
OBJ_TYP_LEAF_LIST	Yes
OBJ_TYP_LIST	Yes
OBJ_TYP_CHOICE	No
OBJ_TYP_CASE	No
OBJ_TYP_USES	No
OBJ_TYP_REFINE	No
OBJ_TYP_AUGMENT	No
OBJ_TYP_RPC	No
OBJ_TYP_RPCIO	No
OBJ_TYP_NOTIF	No

5.2.1 Child Data Nodes

The design used for the storage of child nodes has changed over time.



The old design requires the entire child list to be searched when adding a new entry. The new design uses a queue of object headers. First the correct object is found, then the instance of that object is found. YANG lists are stored in an AVL tree (using libdict hb_tree).

The functions **val_set_canonical_order** and **check_duplicate_list** are very expensive if there are many child nodes for a given parent list or container. These functions are no longer used because the **val_add_child** function maintains YANG schema order and checks for duplicate list entries.

The performance improvements are dramatic. A simple (but expensive) test is to send the server an <edit-config> operation that creates 50,000 list entries in 1 request. Another test is the time it takes the server to boot with these 50,000 list entries.

Description	**OLD Server**	**NEW Server**
Time for <edit-config> with 50,000 list entries	108.27 sec	0.355 sec
Entries per second for <edit-config>	462	140,845
Load 50,000 entries at startup	62.32 sec	0.177 sec
Entries per second for startup	802	282,485

New Modules

- **val_child.c:** child handler code moved from **val.c**
- **val_tree.c** AVL tree support for child nodes

System Changes

1) Entries are no longer sorted. The “system_sorted” parameter is ignored. NETCONF does not require entries to be sorted by index. It only requires the order to be maintained if “ordered-by user” is configured in the YANG module. The server will maintain the order or instances given in the <edit-config> or startup. It will maintain YANG schema order so the objects will be returned in the correct order according to the YANG module.

2) val_set_index_chain MUST be called before val_add_child for a list

Sometimes server or SIL code creates data. In order to create a list, the following steps must be followed

a) Create the list node (E.g., **val_new_value()**)

b) Create the key leafs and add them to the parent with **val_add_child**

c) Create the list index chain with **val_gen_index_chain**

d) Add the list node to its parent with **val_add_child**

Make sure step (d) is done AFTER steps (b) and (c)

If this is not possible, use **val_child_add_force** instead of **val_add_child**

3) There are multiple variants of val_add_child. Sometimes the value is only created so it can be output in XML or JSON, and not to be stored in the database. The functions in val_child should be used instead of **val.c**. The **val.c** functions will still work, but the new functions should be used in new code. The new functions can fail with an error, so it is important to check the return value in server code. The **yangcli-pro** and compiler programs will use **val_child_add_force** by default, since they do not store the data nodes in a database.

Function	Description
val_child_add	Replaces val_add_child and val_add_child_sorted
val_child_add_force	Allows AVL tree insertion to be skipped for duplicates or incomplete list entries
val_insert_child	Replaces val_insert_child

5.2.2 Data Node Types

The **ncx_btype_t** enumeration in **ncx/ncxtypes.h** is used within each **val_value_t** to quickly identify which variant of the data node structure is being used.

The following table describes the different enumeration values:

YumaPro Data Types (ncx_btype_t)

Data Type	Description
NCX_BT_NONE	No type has been set yet. The val_new_value() function has been called but no specific init function has been called to set the base type.
NCX_BT_ANY	The node is a YANG 'anyxml' node. When the client or server parses an 'anyxml' object, it will be converted to containers and strings. This type should not be used directly.
NCX_BT_BITS	YANG 'bits' data type
NCX_BT_ENUM	YANG 'enumeration' data type
NCX_BT_EMPTY	YANG 'empty' data type
NCX_BT_BOOLEAN	YANG 'boolean' data type
NCX_BT_INT8	YANG 'int8' data type
NCX_BT_INT16	YANG 'int16' data type
NCX_BT_INT32	YANG 'int32' data type
NCX_BT_INT64	YANG 'int64' data type
NCX_BT_UINT8	YANG 'uint8' data type
NCX_BT_UINT16	YANG 'uint16' data type
NCX_BT_UINT32	YANG 'uint32' data type
NCX_BT_UINT64	YANG 'uint64' data type
NCX_BT_DECIMAL64	YANG 'decimal64' data type
NCX_BT_FLOAT64	Hidden double type, used just for XPath. If the HAS_FLOAT #define is false, then this type will be implemented as a string, not a double.
NCX_BT_STRING	YANG 'string' type. There are also some YumaPro extensions that are used with this data type for special strings. The server needs to know if a string contains XML prefixes or not, and there are several flavors to automatate processing of each one correctly.
NCX_BT_BINARY	YANG 'binary' data type
NCX_BT_INSTANCE_ID	YANG 'instance-identifier' data type
NCX_BT_UNION	YANG 'union' data type. This is a meta-type. When the client or server parses a value, it will resolve the union to one of the data types defined within the union.
NCX_BT_LEAFREF	YANG 'leafref' data type. This is a meta-type. The client or server will resolve this data type to the type of the actual 'pointed-at' leaf that is being referenced.
NCX_BT_IDREF	YANG 'identityref' data type

Data Type	Description
NCX_BT_SLIST	XSD list data type (ncx:xsdlist extension)
NCX_BT_CONTAINER	YANG container
NCX_BT_CHOICE	YANG choice. This is a meta-type and placeholder. It does not appear in the data tree.
NCX_BT_CASE	YANG case. This is a meta-type and placeholder. It does not appear in the data tree.
NCX_BT_LIST	YANG list
NCX_BT_EXTERN	Internal 'external' data type, used in yangcli-pro. It indicates that the content is actually in an external file.
NCX_BT_INTERN	Internal 'buffer' data type, used in yangcli-pro. The content is actually stored verbatim in an internal buffer.

5.2.3 YumaPro Data Node Edit Variables (val_editvars_t)

There is a temporary data structure which is attached to a data node while editing operations are in progress, called **val_editvars_t**. This structure is used by the functions in **agt/agt_val.c** to manipulate the value tree nodes during an **<edit-config>**, **<copy-config>**, **<load-config>**, or **<commit>** operation.

The SIL callback functions may wish to refer to the fields in this data structure. There is also a SIL cookie field to allow data to be transferred from one callback stage to the later stages. For example, if an edit operation caused the device instrumentation to reserve some memory, then this cookie could store that pointer.

The following typedef is used to define the **val_editvars_t** structure:

```
/* one set of edit-in-progress variables for one value node */
typedef struct val_editvars_t_ {
    /* these fields are only used in modified values before they are
     * actually added to the config database (TBD: move into struct)
     * curparent == parent of curnode for merge
     */
    struct val_value_t_  *curparent;
    op_editop_t    editop;            /* effective edit operation */
    op_insertop_t  insertop;             /* YANG insert operation */
    xmlChar       *insertstr;          /* saved value or key attr */
    struct xpath_pcb_t_ *insertxpcb;       /* key attr for insert */
    struct val_value_t_ *insertval;                    /* back-ptr */
    boolean        iskey;                    /* T: key, F: value */
    boolean        operset;                 /* nc:operation here */
    void          *pcookie;                /* user pointer cookie */
    int            icookie;                /* user integer cookie */
} val_editvars_t;
```

The following fields within the **val_editvars_t** are highlighted:

val_editvars_t Fields

Field	Description
curparent	A 'new' node will use this field to remember the parent of the 'current' value. This is needed to support the YANG insert operation.
editop	The effective edit operation for this node.
insertop	The YANG insert operation, if any.
insertstr	The YANG 'value' or 'key' attribute value string, used to support the YANG insert operation.
insertxpcb	XPath parser control block for the insert 'key' expression, if needed. Used to support the YANG insert operation.
insertval	Back pointer to the value node to insert ahead of, or behind, if needed. Used to support the 'before' and 'after' modes of the YANG insert operation.
iskey	TRUE if this is a key leaf. FALSE otherwise.
operset	TRUE if there was an nc:operation attribute found in this node; FALSE if the 'editop' is derived from its parent.

Field	Description
pcookie	SIL user pointer cookie. Not used by the server. Reserved for SIL callback code.
icookie	SIL user integer cookie. Not used by the server. Reserved for SIL callback code.

5.2.4 YumaPro Data Nodes (val_value_t)

The **val_value_t** data structure is used to maintain the internal representation of all NETCONF databases, non-configuration data available with the <get> operation, all RPC operation input and output parameters, and all notification contents.

The following typedef is used to define a value node:

```
/* one value to match one type */
typedef struct val_value_t_ {
    dlq_hdr_t      qhdr;

    /* common fields */
    struct obj_template_t_ *obj;            /* bptr to object def */
    typ_def_t *typdef;                   /* bptr to typdef if leaf */
    const xmlChar *name;                  /* back pointer to elname */

    /* the dname field is moved to val_extra_t and only used when
     * the value is constructed from dummy objects or no objects at all
     */

    struct val_value_t_ *parent;       /* back-ptr to parent if any */
    struct val_child_hdr_t_ *hdr;      /* back-ptr to own child_hdr */

    val_extra_t *val_extra;

    uint32         flags;                 /* internal status flags */

    xmlns_id_t     nsid;              /* namespace ID for this node */
    ncx_btype_t    btyp;                 /* base type of this value */
    ncx_data_class_t dataclass;             /* config or state data */

    /* last_modified and etag fields used for filtered retrieval
     * and YANG-API If-Match type of conditional editing     */
    time_t         last_modified;
    ncx_etag_t     etag;

    /* YANG does not support user-defined meta-data but NCX does.
     * The <edit-config>, <get> and <get-config> operations
     * use attributes in the RPC parameters, the metaQ is still used
     *
     * The ncx:metadata extension allows optional attributes
     * to be added to object nodes for anyxml, leaf, leaf-list,
     * list, and container nodes.  The config property will
     * be inherited from the object that contains the metadata
     *
     * This is used mostly for RPC input parameters
     * and is strongly discouraged.  Full edit-config
     * support is not provided for metadata
     */
    dlq_hdr_t        *metaQ;                    /* Q of val_value_t */

    /* value editing variables
     * the editvars will only be malloced while edit is in progress
     */
    val_editvars_t  *editvars;              /* edit-vars from attrs */
    op_editop_t      editop;                /* needed for all edits */
    status_t         res;                      /* validation result */
```

```
    /* GET1 getcb moved to val_extra */
    /* GET1 virtualval moved to val_extra */
    /* GET1 cachetime moved to val_extra */

    /* this field used in NCX_BT_LIST only ??? when is it ncx_filptr_t ??? */
    dlq_hdr_t  *indexQ;    /* Q of val_index_t or ncx_filptr_t */

    /* this field is used for NCX_BT_CHOICE
     * If set, the object path for this node is really:
     *    $this --> casobj --> casobj.parent --> $this.parent
     * the OBJ_TYP_CASE and OBJ_TYP_CHOICE nodes are skipped
     * inside an XML instance document
     *
     * replaced by val_get_casobj() function in val_util.h
     * struct obj_template_t_   *casobj;
     */

    /* xpathpcb moved to val_extra_t */

    /* back-ptr to the partial locks that are held
     * against this node
     * plock moved to val_extra
     */

    /* back-ptr to the data access control rules that
     * reference this node
     * dataruleQ moved to val_extra
     */

    /* malloced pointer to the variable expression found
     * if this val node is part of a data template.
     * The actual value in union v_ MUST be ignored if
     * varexpr string is non-NULL!!
     * varexpr moved to val_extra
     */

    /* set if the owners are being stored and this is
     * a data node in the running config
     */
    ncx_owner_id_t owner_id;

    /* NMDA origin enum */
    ncx_nmda_origin_t nmda_origin;

    /* union of all the NCX-specific sub-types
     * note that the following invisible constructs should
     * never show up in this struct:
     *     NCX_BT_CHOICE
     *     NCX_BT_CASE
     *     NCX_BT_UNION
     */
    union v_ {
       /* complex types have a Q of val_value_t representing
        * the child nodes with values
        *   NCX_BT_CONTAINER
        *   NCX_BT_LIST
          *
          * changed from Q of val_value_t to a Q of val_child_hdr_t
        */
        dlq_hdr_t   child_hdrQ;

        /* Numeric data types:
        *   NCX_BT_INT8, NCX_BT_INT16,
```

```
       *   NCX_BT_INT32, NCX_BT_INT64
       *   NCX_BT_UINT8, NCX_BT_UINT16
       *   NCX_BT_UINT32, NCX_BT_UINT64
       *   NCX_BT_DECIMAL64, NCX_BT_FLOAT64
       */
      ncx_num_t   num;

      /* String data types:
       *   NCX_BT_STRING
       *   NCX_BT_INSTANCE_ID
       */
      ncx_str_t  str;

      val_idref_t idref;                /* NCX_BT_IDREF */
      ncx_binary_t binary;             /* NCX_BT_BINARY */
      ncx_list_t list;      /* NCX_BT_BITS, NCX_BT_SLIST */
      boolean    boo;    /* NCX_BT_EMPTY, NCX_BT_BOOLEAN */
      ncx_enum_t enu;       /* NCX_BT_UNION, NCX_BT_ENUM */
      xmlChar   *fname;                /* NCX_BT_EXTERN */
      xmlChar   *intbuff;              /* NCX_BT_INTERN */
    } YPACK v;
} YPACK val_value_t;
```

The following table highlights the fields in this data structure.

Note: Do not access fields directly! Use val.h macros or access functions in val.h and val_util.h

val_value_t Fields

Field	Description
qhdr	Internal queue header to allow a value node to be stored in a queue. A complex node maintains a child queue of val_value_t nodes.
obj	Back pointer to the object template for this data node
typdef	Back pointer to the typedef structure if this is a leaf or leaf-list node.
name	Back pointer to the name string for this node
parent	Back pointer to the parent of this node, if any
nsid	Namespace ID for this node. This may not be the same as the object node namespace ID, E.g., anyxml child node contents will override the generic object namespace.
btyp	The ncx_btype_t base type enumeration for this node. This is the final resolved value, in the event the object type is not a final resolved base type.
flags	Internal flags field. Do not access directly.
dataclass	Internal config or non-config enumeration
metaQ	Queue of val_value_t structures that represent any meta-variables (XML attributes) found for this data node. For example, the NETCONF filter 'type' and 'select' attributes are defined for the

Field	Description
	<filter> element in yuma-netconf.yang.
editvars	Pointer to the malloced edit variables structure for this data node. This node will be freed (and NULL value) when the edit variables are not in use.
res	Internal validation result status for this node during editing or parsing.
indexQ	Queue of internal data structures used during parsing and filtering streamed output.
val_extra	Pointer to extra data if needed
v	Union of different internal fields, depending on the 'btyp' field value.
v.childQ	Queue of val_value_t child nodes, if this is a complex node.
v.num	ncx_num_t for all numeric data types
v.str	Malloced string value for the string data type
v.idref	Internal data structure for the YANG identityref data type
v.binary	Internal data structure for the YANG binary data type
v.list	Internal data structure for YANG bits and NCX xsdlist data types
v.boo	YANG boolean data type
v.enu	Internal data structure for YANG enumeration data type
v.fname	File name for NCX 'external' data type
v.intbuff	Malloced buffer for 'internal' data type

5.2.5 YumaPro Data Nodes (Extra) (val_extra_t)

The **val_extra_t** data structure is used to maintain the fields that are internal and rarely used by the server have been moved from **val_value_t** to a new data structure called **val_extra_t**.

The following typedef is used to define the extra data used within a **val_value_t** value node:

```
/* extra information not used very often within a val_value_t */
typedef struct val_extra_t_ {

    /* malloced value name used rarely if obj_get_name not correct
     * this can happen if val_value_t trees are constructed by SIL code
     * from generic objects;  This is NOT RECOMMENDED
     * Use val_init_from_template using correct obj_template_t instead
     */
    xmlChar         *dname;          /* AND malloced name if needed */

    /* Used by Agent only: GET1 callback for virtualval
     * if this field is non-NULL, then the entire value node
     * is actually a placeholder for a dynamic read-only object
     * and all read access is done via this callback function;
     * the real data type is getcb_fn_t *
     */
    void *getcb;

    /* if this field is non-NULL, then a malloced value struct
     * representing the real value retrieved by
     * val_get_virtual_value, is cached here for <get>/<get-config>
     */
    struct val_value_t_ *virtualval;
    time_t               cachetime;

    /* these fields are for NCX_BT_LEAFREF
     * NCX_BT_INSTANCE_ID, or tagged ncx:xpath
     * value stored in v union as a string
     */
    struct xpath_pcb_t_            *xpathpcb;


    /* back-ptr to the partial locks that are held
     * against this node
     */
    plock_cb_t  *plock[VAL_MAX_PLOCKS];


    /* back-ptr to the data access control rules that
     * reference this node
     */
    dlq_hdr_t *dataruleQ;   /* Q of obj_xpath_ptr_t */

    /* malloced pointer to the variable expression found
     * if this val node is part of a data template.
     * The actual value in union v_ MUST be ignored if
     * varexpr string is non-NULL!!
     */
    xmlChar *varexpr;

} YPACK val_extra_t;
```

The following table highlights the fields in this data structure:

Note: Do not access fields directly! Use val.h macros or access functions in val.h and val_util.h

val_extra_t Fields

Field	Description
dname	Malloced name string if the client or server changed the name of this node, so the object node name is not being used. This is used for anyxml processing (and other things) to allow generic objects (container, string, empty, etc.) to be used to represent the contents of an 'anyxml' node.
getcb	Internal server callback function pointer. Used only if this is a 'virtual' node, and the actual value node contents are generated by a SIL callback function instead of being stored in the node itself.
virtualval	The temporary cached virtual node value, if the getcb pointer is non-NULL.
cachetime	The timestamp used to cache tohe virtualval
xpathpcb	XPath parser control block, used if this value contains some sort of XPath string or instance-identifier. For example, the XML namespace ID mappings are stored, so the XML prefix map generated for the <rpc-reply> will contain and reuse the proper namespace attributes, as needed.
plock	Array of [1] for supporting <partial-lock> operation
dataruleQ	pointer to malloced queue for NACM data rule support
varexpr	malloced string containing variable string for replacement

5.2.6 val_value_t Access Macros

There are a set of macros defined to access the fields within a **val_value_t** structure.

These should be used instead of accessing the fields directly. There are also functions defined as well. These macros are provided in addition the the access functions for quick access to the actual node value. These macros must only be used when the base type ('btyp') field has been properly set and known by the SIL code. Some auto-generated SIL code uses these macros.

The following table summarized the **val_value_t** macros that are defined in **ncx/val.h**:

Macro	**Description**
VAL_OBJ(V)	Access object template of the value
VAL_TYPE(V)	Access base type of the value
VAL_HDR(V)	Access child header pointer of the value
VAL_TYPDEF(V)	Access type definition of the value
VAL_NSID(V)	Access namespace ID value
VAL_NAME(V)	Access name value
VAL_RES(V)	Access resource enumeration value
VAL_BOOL(V)	Access value for NCX_BT_BOOLEAN
VAL_EMPTY(V)	Access value for NCX_BT_EMPTY
VAL_DOUBLE(V)	Access value for NCX_BT_FLOAT64
VAL_STRING(V)	Access value for NCX_BT_STRING
VAL_BINARY(V)	Access value for NCX_BT_BINARY
VAL_ENU(V)	Access entire **ncx_enum_t** structure for NCX_BT_ENUM
VAL_ENUM(V)	Access enumeration integer value for NCX_BT_ENUM
VAL_ENUM_NAME(V)	Access enumeration name string for NCX_BT_ENUM
VAL_FLAG(V)	Deprecated: use VAL_BOOL instead
VAL_LONG(V)	Access NCX_BT_INT64 value
VAL_INT(V)	Access NCX_BT_INT32 value
VAL_INT8(V)	Access NCX_BT_INT8 value
VAL_INT16(V)	Access NCX_BT_INT16 value
VAL_INT32(V)	Access NCX_BT_INT32 value
VAL_INT64(V)	Access NCX_BT_INT64 value
VAL_STR(V)	Deprecated: use VAL_STRING instead
VAL_INSTANCE_ID(V)	Access NCX_BT_INSTANCE_ID value
VAL_IDREF(V)	Access entire **val_idref_t** structure for NCX_BT_IDREF
VAL_IDREF_NSID(V)	Access the identityref namespace ID for NCX_BT_IDREF
VAL_IDREF_NAME(V)	Access the identityref name string for NCX_BT_IDREF

Macro	Description
VAL_UINT(V)	Access the NCX_BT_UINT32 value
VAL_UINT8(V)	Access the NCX_BT_UINT8 value
VAL_UINT16(V)	Access the NCX_BT_UINT16 value
VAL_UINT32(V)	Access the NCX_BT_UINT32 value
VAL_UINT64(V)	Access the NCX_BT_UINT64 value
VAL_ULONG(V)	Access the NCX_BT_UINT64 value
VAL_DEC64(V)	Access the **ncx_dec64** structure for NCX_BT_DEC64
VAL_DEC64_DIGITS(V)	Access the number of digits for NCX_BT_DEC64
VAL_DEC64_ZEROES(V)	Access the number of zeros for NCX_BT_DEC64
VAL_LIST(V)	Access the **ncx_list_t** structure for NCX_BT_LIST
VAL_BITS	Access the **ncx_list_t** structure for NCX_BT_BITS. (Same as VAL_LIST)

5.2.7 val_value_t Access Functions

The file **ncx/val.h** contains many API functions so that object properties do not have to be accessed directly. In addition, the file **ncx/val_util.h** contains more (high-level) utility functions. The following table highlights the most commonly used functions. Refer to the H files for a complete definition of each API function.

val_value_t Access Functions

Function	Description
val_new_value	Malloc a new value node with type NCX_BT_NONE.
val_init_complex	Initialize a malloced value node as one of the complex data types.
val_init_virtual	Initialize a malloced value node as a virtual node (provide a 'get' callback function).
val_init_from_template	Initialize a malloced value node using an object template. This is the most common form of the init function used by SIL callback functions.
val_free_value	Clean and free a malloced value node.
val_set_name	Set or replace the value node name.
val_force_dname	Set (or reset) the name of a value struct . Set all descendant nodes as well Force dname to be used, not object name backptr.
val_set_qname	Set or replace the value node namespace ID and name.
val_string_ok	Check if the string value is valid for the value node object type.
val_string_ok_errinfo	Check if the string value is valid for the value node object type, and provide the error information to use if it is not OK.
val_binary_ok_errinfo	Retrieve the YANG custom error info if any. Check a binary string to make sure the value is valid base64 if the value came from raw RPC XML, and if its CLI input then do validation as for binary sting . Retrieve the configured error info struct if any error.
val_list_ok	Check if the list value is valid for the value node object type.
val_list_ok_errinfo	Check if the list value is valid for the value node object type, and provide the error information to use if it is not OK.
val_enum_ok	Check if the enumeration value is valid for the value node object type.
val_enum_ok_errinfo	Check if the enumeration value is valid for the value node object type, and provide the error information to use if it is not OK.
val_bit_ok	Check if the bits value is valid for the value node object type.
val_idref_ok	Check if the identityref value is valid for the value node object type.
val_parse_idref	Convert a string to an internal QName string into its various parts and find the identity struct that is being referenced (if available).
val_range_ok	Check a number to see if it is in range or not . Could be a number or size range.
val_range_ok_errinfo	Check a number to see if it is in range or not . Could be a number or size range. Returns error info record on error exit.

Function	Description
val_pattern_ok	Check a string against all the patterns in a big AND expression.
val_pattern_ok_errinfo	Check a string against all the patterns in a big AND expression. Returns error info record on error exit.
val_simval_ok	Check if the smple value is valid for the value node object type.
val_simval_ok_errinfo	Check if the simple value is valid for the value node object type, and provide the error information to use if it is not OK.
val_union_ok	Check a union to make sure the string is valid.
val_union_ok_errinfo	Check a union to make sure the string is valid. Returns error info record on error exit.
val_union_ok_ex	Check a union to make sure the string is valid based on the specified typdef.
val_union_ok_full	Check a union to make sure the string is valid based on the specified typdef, and convert the string to an NCX internal format . Use context and root nodes to evaluate XPath for leafref and instance-identifier types.
val_get_metaQ	Get the meta Q header for the value.
val_get_first_meta	Get the first meta-variable (XML attribute value) for a value node.
val_get_next_meta	Get the next meta-variable (XML attribute value) for a value node.
val_meta_empty	Check if the metaQ is empty for the value node.
val_find_meta	Find the specified meta-variable in a value node.
val_meta_match	Return true if the corresponding attribute exists and has the same value.
val_metadata_inst_count	Get the number of instances of the specified attribute.
val_dump_value	Debug function to print the contents of any value node.
val_dump_value_ex	Debug function to print the contents of any value node, with extended parameters to control the output.
val_dump_value_max	Debug function to print the contents of any value node, with full control of the output parameters.
val_set_string	Set a malloced value node as a generic string value. Used instead of **val_init_from_template.**
val_set_string2	Set a malloced value node as a specified string type. Used instead of **val_init_from_template**.
val_set_binary	Set and decode base64 value . Set an initialized **val_value_t** as a binary type.
val_reset_empty	Recast an already initialized value as an NCX_BT_EMPTY clean a value and set it to empty type used by **yangcli** to delete leafs.
val_set_simval	Set a malloced value node as a specified simple type. Used instead of **val_init_from_template.**
val_make_simval	Create and set a **val_value_t** as a simple type same as **val_set_simval,** but malloc the value first.
val_set_simval_str	Set a malloced value node as a specified simple type. Used instead of

Function	Description
	val_init_from_template. Use a counted string value instead of a zero-terminated string value.
val_make_string	Create a complete malloced generic string value node.
val_merge	Merge source val into destination value (! MUST be same type !) . Any meta vars in source are also merged into destination . This function is not used to merge complex objects !!! For **typ_is_simple()** only !!!
val_clone	Clone a value node.
val_clone2	Clone a specified **val_value_t** structure and sub-trees but not the editvars.
val_clone_test	Clone a value node with a 'test' callback function to prune certain descendant nodes during the clone procedure.
val_clone_config_data	Clone a value node but skip all the non-configuration descendant nodes.
val_replace	Replace a specified val_value_t structure and sub-trees !!! this can be destructive to the source 'val' parameter !!!!
val_replace_str	Replace a specified **val_value_t** structure with a string type.
val_fast_replace_string	Replace a specified **val_value_t** structure with a string type . Reuse everything -- just set the val->v.str field
val_add_child	Add a child value node to a parent value node. Simply makes a new last child!!! Does not check siblings!!! Relies on val_set_canonical_order To modify existing extries, use val_add_child_sorted instead!!
val_add_child2	Add a child value node to a parent value node.
val_add_child_sorted	Add a child value node to a parent value node in the proper place
val_insert_child	Insert a child value node into a specific spot into a parent value node.
val_remove_child	Remove a child value node from its parent.
val_swap_child	Replace a child node within its parent with a different value node.
val_first_child_match	Match a child node name; Used for partial command completion in **yangcli-pro.**
val_first_child_match_fast	Get the first instance of the corresponding child node. Object pointers must be from the same tree!!!
val_next_child_match	Match the next child node name; Used for partial command completion in **yangcli-pro**.
val_next_child_same	Get the next instance of the corresponding child node.
val_get_first_child	Get the first child value node.
val_get_next_child	Get the next child value node.
val_find_child	Find a specific child value node.
val_find_child_fast	Find the first instance of the specified child node.
val_find_child_obj	Find the first instance of the specified child node . Use the object

Function	Description
	template pointer to match the object.
val_find_child_que	Find the first instance of the specified child node in the specified child Q.
val_find_next_child	Find the next occurrence of a specified child node.
val_find_next_child_fast	Find the next instance of the specified child node . Use the curchild->obj pointer to find next child of this type . Assumes childQ is sorted and all instances of 'curchild' are already grouped together.
val_first_child_name	Get the first corresponding child node instance, by name find first -- really for resolve index function.
val_first_child_qname	Get the first corresponding child node instance, by Qname.
val_next_child_qname	Get the next corresponding child node instance, by Qname.
val_first_child_string	Find first name value pair . Get the first corresponding child node instance, by name and by string value. Child node must be a base type of NCX_BT_STRING NCX_BT_INSTANCE_ID NCX_BT_LEAFREF
val_get_first_terminal_child	Get the child node only if **obj_is_terminal(obj)** is true.
val_get_next_terminal_child	Get the next child node only if **obj_is_terminal(obj)** is true.
val_match_child	Match a potential partial node name against the child node names, and return the first match found, if any.
val_match_child_count	Match the first instance of the specified child node . Return the total number of matches.
val_child_cnt	Get the number of child nodes within a parent node.
val_child_inst_cnt	Get the corresponding child instance count by name . Get instance count -- for instance qualifier checking.
val_get_child_inst_id	Get the instance ID for this child node within the parent context.
val_liststr_count	Get the number of strings within an NCX_BT_LIST value node.
val_index_match	Check if 2 value list nodes have the same set of key leaf values.
val_compare	Compare 2 value nodes
val_compare_ex	Compare 2 value nodes with extra parameters.
val_compare_to_string	Compare a value node to a string value instead of another value node.
val_compare_to_string_len	Compare a **val_value_t** structure value contents to a string . Provide a max-length to compare.
val_sprintf_simval_nc	Output the value node as a string into the specified buffer.
val_make_sprintf_string	Malloc a buffer and fill it with a zero-terminated string representation of the value node.
val_resolve_scoped_name	Find a descendant node within a value node, from a relative path expression.
val_has_content	Return TRUE if the value node has any content; FALSE if an empty XML element could represent its value.

Function	Description
val_has_index	Return TRUE if the value node is a list with a key statement.
val_get_first_index	Get the first index node for the specified list value node.
val_get_next_index	Get the next index node for the specified list value node.
val_get_index_count	Get the number of index nodes in this val.
val_set_extern	Set a malloced value node as an NCX_BT_EXTERN internal data type.
val_set_intern	Set a malloced value node as an NCX_BT_INTERN internal data type.
val_fit_oneline	Return TRUE if the value node should fit on 1 display line; Sometimes a guess is made instead of determining the exact value. XML namespace declarations generated during XML output can cause this function value to sometimes be wrong.
val_create_allowed	Return TRUE if the NETCONF create operation is allowed for the specified value node.
val_delete_allowed	Return TRUE if the NETCONF delete operation is allowed for the specified value node.
val_is_config_data	Return TRUE if the value node represents configuration data.
val_is_virtual	Check if the specified value is a virtual value , such that a 'get' callback function is required to access the real value contents.
val_get_virtual_value	Get the value for a virtual node from its 'get' callback function.
val_is_default	Return TRUE if the value node is set to its YANG default value.
val_is_real	Check if a value node is a real node or one of the abstract node types.
val_get_parent_nsid	Get the namespace ID for the parent value node of a specified child node.
val_instance_count	Get the number of occurrences of the specified child value node within a parent value node.
val_instance_count_fast	Count the number of instances of the specified object name in the parent value structure. This only checks the first level under the parent, not the entire subtree.
val_instance_count_fast2	Count the number of instances of the specified object name in the parent value structure. This only checks the first level under the parent, not the entire subtree . Starts with the startval passed in.
val_need_quotes	Return TRUE if the printed string representation of a value node needs quotes (because it contains some whitespace or special characters).
val_all_whitespace	Check if a string is all whitespace.
val_any_whitespace	Check if a string has any whitespace chars.
val_get_dirty_flag	Check if a value node has been altered by an RPC operation, but this edit has not been finalized yet.
val_get_nest_level	Get the current numeric nest level of the specified value node.
val_get_mod_name	Get the module name for the specified value node.
val_get_mod_prefix	Get the module prefix string for the specified value node.

Function	Description
val_get_nsid	Get the namespace ID for the specified value node.
val_change_nsid	Change the namespace ID for the specified value node and all of its descendants.
val_set_pcookie	Set the SIL pointer cookie in the value node editvars structure.
val_set_icookie	Set the SIL integer cookie in the value node editvars structure.
val_get_pcookie	Get the SIL pointer cookie in the value node editvars structure.
val_get_icookie	Get the SIL integer cookie in the value node editvars structure.
val_get_typdef	Get the typedef structure for a leaf or leaf-list value node.
val_move_children	Move all the child nodes of one complex value node to another complex value node.
val_set_canonical_order	Re-order the descendant nodes of a value node so they are in YANG order. Does not change the relative order of system-ordered lists and leaf-lists.
val_gen_index_chain	Generate the internal key leaf lookup chain for a list value node..
val_add_defaults	Generate the leafs that have default values.
val_instance_check	Check a value node against its template to see if the correct number of descendant nodes are present.
val_get_choice_first_set	Get the first real node that is present for a conceptual choice statement.
val_get_choice_next_set	Get the next real node that is present for a conceptual choice statement.
val_choice_is_set	Return TRUE if some real data node is present for a conceptual choice statement.
val_get_first_key	Get the first key record if this is a list with a key-stmt.
val_get_last_key	Get the last key record if this is a list with a key-stmt.
val_get_next_key	Get the next key record if this is a list with a key-stmt.
val_get_prev_key	Get the previous key record if this is a list with a key-stmt.
val_remove_key	Remove a key pointer because the key is invalid . Free the key pointer.
val_new_child_val	Create a child node during an edit operation. Used by the server. SIL code does not need to maintain the value tree.
val_gen_instance_id	Malloc and generate the YANG instance-identifier string for the value node.
val_check_obj_when	Check if an object node has any 'when' statements, and if so, evaluate the XPath condition(s) against the value tree to determine if the object should be considered present or not.
val_check_child_conditional	Check if a child object node has any FALSE 'if-feature' or 'when' statements.
val_is_mandatory	Check if the child object node is currently mandatory or optional.
val_get_xpathpcb	Access the XPath parser control block for this value node, if any.
val_make_simval_obj	Malloc and fill in a value node from an object template and a value string.

Function	Description
val_set_simval_obj	Fill in a value node from an object template and a value string.
val_find_bit	Check if a bits value contains the specified bit name
val_find_bit_name	Get the name of the bit from a bits value that corresponds to the specified bit position
val_find_enum_name	Get the name of the enum from an enumeration value that corresponds to the specified 'value' number
val_sprintf_etag	Write the Entity Tag for the value to the specified buffer.
val_get_last_modified	Get the last_modified field.
val_has_children	Determine if there are any child nodes for this val.
val_delete_children	Check if the value is a complex type and if so then delete all child nodes.
val_clean_value	Clean a static **val_value_t** structure.
val_get_yang_typename	Get the YANG type name for this value node if there is one.
val_is_value_set	Check if a value has been set by a client . It has to be initialized and not set by default to return true.
val_idref_derived_from	Check the specified valnode is derived from the specified identity.
val_get_sil_priority	Get the secondary SIL priority; zero if not found.
val_has_conditional_value	YANG 1.1: Check the value that must be properly set to see if it is conditional on any if-feature-stmts.
val_convert_leafref	Convert a value of type NCX_BT_LEAFREF to the value that the final leafref is pointing at.

5.2.8 SIL Utility Functions

There are some high-level SIL callback utilities in **agt/agt_util.h**. These functions access the lower-level functions in **libncx** to provide simpler functions for common SIL tasks.

The following table highlights the functions available in this module:

agt/agt_util Functions

Function	Description
agt_get_cfg_from_parm	For value nodes that represent a NETCONF configuration database name (E.g., empty element named 'running'). The configuration control block for the referenced database is retrieved.
agt_get_inline_cfg_from_parm	For value nodes that represent inline NETCONF configuration data. The value node for the inline config node is retrieved.
agt_get_parmval	Get the specified parameter name within the RPC input section, from an RPC message control block.
agt_record_error	Generate a complete RPC error record to be used when the <rpc-reply> is sent.
agt_record_error_errinfo	Generate a complete RPC error record to be used when the <rpc-reply> is sent, using the YANG specified error information, not the default error information.
agt_record_warning	Generate an **rpc_err_rec_t** and save it in the msg . Use the provided error info record for <rpc-error> fields . Set the error-severity field to warning instead of error , but only if agt_with_warnings is TRUE
agt_record_attr_error	Generate a complete RPC error record to be used when the <rpc-reply> is sent for an XML attribute error.
agt_record_insert_error	Generate a complete RPC error record to be used when the <rpc-reply> is sent for a YANG insert operation error.
agt_record_unique_error	Generate a complete RPC error record to be used when the <rpc-reply> is sent for a YANG unique statement constraint error.
agt_validate_filter	Validate the <filter> parameter if present.
agt_check_config	**val_nodetest_fn_t** callback function. Used by the <get-config> operation to return any type of configuration data.
agt_check_default	**val_nodetest_fn_t** Node Test Callback function to filter out default data from streamed replies, according to the server's definition of a default node.
agt_check_config_false	**val_nodetest_fn_t** callback function. Used by the <get*> operation to return only config=false nodes and the ID ancestor nodes.
agt_check_config_false_default	**val_nodetest_fn_t** callback function. Used by the <get> operation to return only content=nonconfig nodes and the ID ancestor nodes and only values not set to the default, except if it is a container with counters.

Function	Description
agt_check_save	**val_nodetest_fn_t** Node Test Callback function to filter out data nodes that should not be saved to NV-storage.
agt_check_max_access	Check if the max-access for a parameter is exceeded.
agt_enable_feature	Enable the specified YANG feature
agt_disable_feature	Disable the specified YANG feature
agt_make_leaf	Create a child value node.
agt_make_uint_leaf	Create a child value node with uint YANG type.
agt_make_int_leaf	Create a child value node with int YANG type.
agt_make_uint64_leaf	Create a child value node with uint64 YANG type.
agt_make_int64_leaf	Create a child value node with int64 YANG type.
agt_make_idref_leaf	Create a child value node with idref YANG type.
agt_make_union_leaf	Create a child value node with union YANG type.
agt_make_bits_leaf	Create a child value node with bit YANG type.
agt_make_boolean_leaf	Create a child value node with boolean YANG type.
agt_make_empty_leaf	Create a child value node with empty YANG type.
agt_make_list	Create a value node for a specified list.
agt_make_object	Create a value node from a specified object template.
agt_make_virtual_leaf	Create a virtual value child node. Most device monitoring leafs use this function because the value is retrieved with a device-specific API, not stored in the value tree.
agt_init_cache	Initialize a cached pointer to a node in a data tree.
agt_check_cache	Check if a cached pointer to a node in a data tree needs to be updated or set to NULL.
agt_get_key_value	Get the next expected key value in the ancestor chain . Used in Yuma SIL code to invoke User SIL callbacks with key values.
agt_any_operations_set	Check the new node and all descendants for any operation attributes present.
agt_match_etag	Check if the etag matches the specified value string.
agt_match_etag_binary	Check if the etag matches the specified value string . Start with a binary etag, not a string.
agt_modified_since	Check if the time-stamp for the object is later then the specified time-stamp.
agt_notifications_enabled	Check if notifications are enabled.
agt_check_delete_all_allowed	Check if the delete-all or remove-all operation is enabled for for specified object.

6 SIL External Interface

Each SIL has 2 initialization functions and 1 cleanup function that must be present.

- The first initialization callback function is used to set up the configuration related objects.
- The second initialization callback is used to setup up non-configuration objects, after the running configuration has been loaded from the startup file.
- The cleanup callback is used to remove all SIL data structures and unregister all callback functions.

These are the only SIL functions that the server will invoke directly. They are generated by **yangdump-pro** with the **--format** parameter, and usually do not require any editing by the developer.

Most of the work done by SIL code is through callback functions for specific RPC operations and database objects. These callback functions are usually registered during the initialization functions.

6.1 Stage 1 Initialization

The stage 1 initialization function is the first function called in the library by the server.

If the **netconfd-pro** configuration parameters include a 'load' command for the module, then this function will be called during server initialization. It can also be called if the **<load>** operation is invoked during server operation.

This function MUST NOT attempt to access any database. There will not be any configuration databases if this function is called during server initialization. Use the 'init2' function to adjust the running configuration.

This callback function is expected to perform the following functions:

- initialize any module static data
- make sure the requested module name and optional revision date parameters are correct
- load the requested module name and revision with **ncxmod_load_module**
- setup top-level object cache pointers (if needed)
- register any RPC method callbacks with **agt_rpc_register_method**
- register any database object callbacks with **agt_cb_register_callback**
- perform any device-specific and/or module-specific initialization

Name Format:

y_<modname>_init

Input:

- modname == string containing module name to load
- revision == string containing revision date to use
  == NULL if the operator did not specify a revision.

Returns:

- operation status (0 if success)

Example function generated by **yangdump-pro**:

```
/********************************************************************
* FUNCTION y_toaster_init
*
* initialize the toaster server instrumentation library
*
* INPUTS:
*    modname == requested module name
*    revision == requested version (NULL for any)
*
* RETURNS:
*     error status
********************************************************************/
status_t
    y_toaster_init (
        const xmlChar *modname,
        const xmlChar *revision)
{
    agt_profile_t *agt_profile;
    status_t res;

    y_toaster_init_static_vars();

    /* change if custom handling done */
    if (xml_strcmp(modname, y_toaster_M_toaster)) {
        return ERR_NCX_UNKNOWN_MODULE;
    }

    if (revision && xml_strcmp(revision, y_toaster_R_toaster)) {
        return ERR_NCX_WRONG_VERSION;
    }

    agt_profile = agt_get_profile();

    res = ncxmod_load_module(
        y_toaster_M_toaster,
        y_toaster_R_toaster,
        &agt_profile->agt_savedevQ,
        &toaster_mod);
    if (res != NO_ERR) {
        return res;
    }

    toaster_obj = ncx_find_object(
        toaster_mod,
        y_toaster_N_toaster);
    if (toaster_mod == NULL) {
        return SET_ERROR(ERR_NCX_DEF_NOT_FOUND);
    }

    make_toast_obj = ncx_find_object(
        toaster_mod,
        y_toaster_N_make_toast);
    if (toaster_mod == NULL) {
        return SET_ERROR(ERR_NCX_DEF_NOT_FOUND);
    }

    cancel_toast_obj = ncx_find_object(
        toaster_mod,
        y_toaster_N_cancel_toast);
    if (toaster_mod == NULL) {
        return SET_ERROR(ERR_NCX_DEF_NOT_FOUND);
    }
```

```
    toastDone_obj = ncx_find_object(
        toaster_mod,
        y_toaster_N_toastDone);
    if (toaster_mod == NULL) {
        return SET_ERROR(ERR_NCX_DEF_NOT_FOUND);
    }

    res = agt_rpc_register_method(
        y_toaster_M_toaster,
        y_toaster_N_make_toast,
        AGT_RPC_PH_VALIDATE,
        y_toaster_make_toast_validate);
    if (res != NO_ERR) {
        return res;
    }

    res = agt_rpc_register_method(
        y_toaster_M_toaster,
        y_toaster_N_make_toast,
        AGT_RPC_PH_INVOKE,
        y_toaster_make_toast_invoke);
    if (res != NO_ERR) {
        return res;
    }

    res = agt_rpc_register_method(
        y_toaster_M_toaster,
        y_toaster_N_cancel_toast,
        AGT_RPC_PH_VALIDATE,
        y_toaster_cancel_toast_validate);
    if (res != NO_ERR) {
        return res;
    }

    res = agt_rpc_register_method(
        y_toaster_M_toaster,
        y_toaster_N_cancel_toast,
        AGT_RPC_PH_INVOKE,
        y_toaster_cancel_toast_invoke);
    if (res != NO_ERR) {
        return res;
    }

    res = agt_cb_register_callback(
        y_toaster_M_toaster,
        (const xmlChar *)"/toaster",
        (const xmlChar *)"2009-11-20",
        y_toaster_toaster_edit);


    if (res != NO_ERR) {
        return res;
    }

    /* put your module initialization code here */

    return res;
} /* y_toaster_init */
```

6.2 Stage 2 Initialization

The stage 2 initialization function is the second function called in the library by the server:

- It will only be called if the stage 1 initialization is called first, and it returns 0 (NO_ERR status).
- This function is used to initialize any needed data structures in the running configuration, such as factory default configuration, read-only counters and status objects.
- It is called after the startup configuration has been loaded into the server.
- If the **<load>** operation is used during server operation, then this function will be called immediately after the state 1 initialization function.

Note that configuration data structures that are loaded during server initialization **(load_running_config**) will be handled by the database callback functions registered during phase 1 initialization.

Any server-created configuration nodes should be created during phase 2 initialization (this function), after examining the explicitly-provided configuration data. For example, the top-level **/nacm** container will be created (by **agt_acm.c**) if it is not provided in the startup configuration.

This callback function is expected to perform the following functions:

- load non-configuration data structures into the server (if needed)
- initialize top-level data node cache pointers (if needed)
- load factory-default configuration data structures into the server (if needed)
- optionally save a cached pointer to a data tree node (such as the root node for the module). The **agt_create_cache** function in **agt/agt_util.h** is used to initialize such a module-static variable.

Name Format:

y_<modname>_init2

Returns:

- operation status (0 if success)

Example function generated by **yangdump-pro**:

```
/********************************************************************
* FUNCTION y_toaster_init2
*
* SIL init phase 2: non-config data structures
* Called after running config is loaded
*
* RETURNS:
*     error status
********************************************************************/

status_t
    y_toaster_init2 (void)
{
    status_t res = NO_ERR;

    toaster_val = agt_init_cache(
        y_toaster_M_toaster,
```

```
        y_toaster_N_toaster,
        &res);
    if (res != NO_ERR) {
        return res;
    }

    /* put your init2 code here */

    return res;
} /* y_toaster_init2 */
```

6.3 Cleanup

The cleanup function is called during server shutdown. It is only called if the stage 1 initialization function is called. It will be called right away if either the stage 1 or stage 2 initialization functions return a non-zero error status.

This callback function is expected to perform the following functions:

- cleanup any module static data
- free any top-level object cache pointers (if needed)
- unregister any RPC method callbacks with **agt_rpc_unregister_method**
- unregister any database object callbacks with **agt_cb_unregister_callbacks**
- perform any device-specific and/or module-specific cleanup

Name Format:

y_<modname>_cleanup

Example function generated by **yangdump-pro**:

```
/********************************************************************
* FUNCTION y_toaster_cleanup
*    cleanup the server instrumentation library
*
*********************************************************************/
void
    y_toaster_cleanup (void)
{
    agt_rpc_unregister_method(
        y_toaster_M_toaster,
        y_toaster_N_make_toast);

    agt_rpc_unregister_method(
        y_toaster_M_toaster,
        y_toaster_N_cancel_toast);



   agt_cb_unregister_callbacks(
        y_toaster_M_toaster,
        (const xmlChar *)"/toaster");

    /* put your cleanup code here */

} /* y_toaster_cleanup */
```

7 SIL Callback Interface

This section briefly describes the SIL code that a developer will need to create to handle the data-model specific details. SIL functions access internal server data structures, either directly or through utility functions. Database mechanics and XML processing are done by the server engine, not the SIL code. A more complete reference can be found in section 5.

When a <rpc> request is received, the NETCONF server engine will perform the following tasks before calling any SIL:

- parse the RPC operation element, and find its associated YANG rpc template
- if found, check if the session is allowed to invoke this RPC operation
- if the RPC is allowed, parse the rest of the XML message, using the **rpc_template_t** for the RPC operation to determine if the basic structure is valid.
- if the basic structure is valid, construct an **rpc_msg_t** data structure for the incoming message.
- check all YANG machine-readable constraints, such as must, when, if-feature, min-elements, etc.
- if the incoming message is completely 'YANG valid', then the server will check for an RPC validate function, and call it if found. This SIL code is only needed if there are additional system constraints to check. For example:
  - need to check if a configuration name such as **<candidate>** is supported
  - need to check if a configuration database is locked by another session
  - need to check description statement constraints not covered by machine-readable constraints
  - need to check if a specific capability or feature is enabled
- If the validate function returns a NO_ERR status value, then the server will call the SIL invoke callback, if it is present. This SIL code should always be present, otherwise the RPC operation will have no real affect on the system.
- At this point, an <rpc-reply> is generated, based on the data in the **rpc_msg_t.**
  - Errors are recorded in a queue when they are detected.
  - The server will handle the error reply generation for all errors it detects.
  - For SIL detected errors, the **agt_record_error** function in **agt/agt_util.h** is usually used to save the error details.
  - Reply data can be generated by the SIL invoke callback function and stored in the **rpc_msg_t** structure.
  - Replay data can be streamed by the SIL code via reply callback functions. For example, the **<get>** and **<get-config>** operations use callback functions to deal with filters, and stream the reply by walking the target data tree.
- After the <rpc-reply> is sent, the server will check for an RPC post reply callback function. This is only needed if the SIL code allocated some per-message data structures. For example, the **rpc_msg_t** contains 2 SIL controlled pointers (**rpc_user1** and **rpc_user2**). The post reply callback is used by the SIL code to free these pointers, if needed.

The database edit SIL callbacks are only used for database operations that alter the database. The validate and invoke callback functions for these operations will in turn invoke the data-model specific SIL callback functions, depending on the success or failure of the edit request.

7.1 Configuration Initialization

During phase 2 initiation the server will load the NV-stored configuration, depending on the CLI and conf file parameters:

- **--config=foo.conf**: The specified configuration file will be loaded. The YUMAPRO_DATAPATH or **--datapath** parameters can be used to control the search path for this XML file.
- **--no-config:** An empty factory-default configuration will be loaded, except the NV-stored version will not be set back to factory default values.
- **--factory-config:** An empty factory-default configuration will be loaded, and the NV-stored version will also be set back to factory default values.
- default: The default configuration file will be loaded. The YUMAPRO_DATAPATH or **--datapath** parameters can be used to control the search path for this XML file (**startup-cfg.xml**)

7.1.1 Validation Phase

Once the initial configuration is parsed and converted to a tree of **val_value_t** structures, it is validated according to the YANG field validation rules in the loaded modules. The SIL edit callback function must not allocate any resources or alter system behavior during the validate phase.

The **--startup-error** CLI or conf file parameter controls how the server proceeds at this point:

- **--startup-error=stop**: Any unknown definitions (namespace, element, attribute) will cause the server to terminate. Any invalid values for the expected data type for each node will cause the server to terminate. This is the default action.
- **--startup-error=continue**: Any unknown definitions (namespace, element, attribute) will cause the server to prune those nodes, log warnings, and continue . Any invalid values for the expected data type for each node will cause the server to prune those nodes, log warnings and continue.

After the configuration is field-validated, the user SIL edit callbacks are called for the validation phase.

The following parameters will be passed to this callback function:

- **scb** == dummy session control block for the system user (called superuser in log output)
- **msg** == dummy RPC request message for the internal <load-config> operation
- **cbtyp** == AGT_CB_VALIDATE
- **editop** == OP_EDITOP_LOAD
- **newval** == The node from the configuration that is being loaded
- **curval** == NULL

After all SIL edit callbacks have been invoked and no errors have been reported, the agt_val_root_check function is run to perform all the YANG datastore validation tests, according to the modules loaded in the server. The steps are enumerated, but actually implemented to be performed at the same time:

- Remove all false when-stmts (delete_dead_nodes)
- Validate that the correct number of instances are present
  - optional container or leaf: 0 or 1 instances
  - mandatory container or leaf: 1 instance
  - mandatory choice: 1 case present
  - list, leaf-list: min-elements, max-elements
- Check YANG specific constraints:
  - list: all keys present
  - list unique-stmt: specified descendant nodes checked across all list entries to make sure no duplicate values in any entries
  - all nodes with must-stmt validation expressions are checked to make sure the Xpath expression result is a boolean with the value 'true'.

7.1.2 Apply Phase

After all validation tests have been run the server decides if it can continue by checking the **--running-error** CLI/conf file parameter:

- **--running-error=stop**: If any errors are reported in the validation phase the server will exit with an error because the running configuration is not valid. This is the default behavior.
- **--running-error=continue**: If any errors are reported in the validation phase the server will attempt to prune the nodes with errors. The server will continue booting even if the configuration is not valid according to the YANG datastore validation rules. The server will remember that the configuration is bad and only perform full validation checks until a valid configuration is saved.

If the server continues beyond this point, then the SIL edit callbacks are all called again for the apply phase. The SIL code can reserve resources at this point but not activate the configuration.

The following parameters will be passed to this callback function:

- **scb** == dummy session control block for the system user (called superuser in log output)
- **msg** == dummy RPC request message for the internal **<load-config>** operation
- **cbtyp** == AGT_CB_APPLY
- **editop** == OP_EDITOP_LOAD
- **newval** == The node from the configuration that is being loaded
- **curval** == NULL

If any SIL callback generates an error during this phase the configuration load will be terminated and the server will shutdown.

7.1.3 Commit/Rollback Phase

If no SIL callback functions generate an error in the apply phase then the server will attempt to commit the configuration. All of the SIL edit callback functions will be called again to commit the configuration. If any SIL callback function generates an error then the server will switch into rollback mode.

The callback type will either be AGT_CB_COMMIT or AGT_CB_ROLLBACK.

The SIL code must activate or free any reserved resources at this point. It will only be called once for either commit or rollback, during the same edit.

If the callback type is AGT_CB_COMMIT then it must also activate the configuration.

If the server attempts to rollback the SIL configuration commits, then any nodes that have already accepted the commit will be called again to validate, apply, and commit a “delete” operation (OP_EDITOP_DELETE) on the data node that was created via the OP_EDIT_LOAD operation.

The following parameters will be passed to this callback function:

- **scb** == dummy session control block for the system user (called superuser in log output)
- **msg** == dummy RPC request message for the internal <load-config> operation
- **cbtyp** == AGT_CB_COMMIT or AGT_CB_ROLLBACK
- **editop** == OP_EDITOP_LOAD
- **newval** == The node from the configuration that is being loaded
- **curval** == NULL

7.2 Configuration Replay

It is possible to replay the entire configuration if the underlying system resets or restarts, but the server process is still running.

7.2.1 Timer Callback to Check System

The configuration replay procedure must be triggered by a timer callback function. This function can be registered in the yp-system library or in a SIL library. The purpose of this callback is to check the health of the underlying system and if it has restarted and needs to be re-initialized with configuration, then the **agt_request_replay()** function must be called.

```
/*********************************************************************
* FUNCTION agt_request_replay
*
* Request replay of the running config to SIL modules
* because SIL hardware has reset somehow
*
*********************************************************************/
extern void agt_request_replay (void);
```

Example Callback Function:

```
/* timer callback function
 *
 * Process the timer expired event
 *
 * INPUTS:
 *    timer_id == timer identifier
 *    cookie == context pointer, such as a session control block,
 *            passed to agt_timer_set function (may be NULL)
 *
 * RETURNS:
 *     0 == normal exit
 *    -1 == error exit, delete timer upon return
 */
static int
    force_replay (uint32  timer_id,
                  void *cookie)
{
    (void)timer_id;
    (void)cookie;

    boolean need_replay = FALSE;

    // check the system somehow

    if (need_replay) {
        agt_request_replay();
    }

    return 0;
```

```
}  /* force_replay */
```

Example Timer Callback Registration

```
status_t u_foo_init (
    const xmlChar *modname,
    const xmlChar *revision)
{
    status_t res = NO_ERR;
    ncx_module_t *foo_mod = NULL;

    foo_mod = ncx_find_module(
        y_foo_M_address,
        y_foo_R_address);
    if (foo_mod == NULL) {
        return ERR_NCX_OPERATION_FAILED;
    }

    /* put your module initialization code here */
    uint32 timerid = 0;

    /* example will check every 10 seconds
    res = agt_timer_create(10, TRUE, force_replay, NULL, &timerid);

    return res;

} /* u_foo_init */
```

7.2.2 SIL Callbacks

The same SIL callback procedure is used for the initial configuration load and a replay config load.

The YANG field and datastore validation is not done. Only the SIL callback functions are called to allow the SIL code to reconfigure the underlying system according to the replay values.

Some parameters are different, and the SIL edit callback functions may need to know the difference, since data structures may already be setup and the SIL code would leak memory if pointers to malloced data were re-initialized without cleaning up first.

If the callback is for a replay then the following macro from **ncx/rpc_msg.h** will return TRUE:

- RPC_MSG_IS_REPLAY(msg):
  - Evaluates to true if the msg is for the <replay-config> operation
  - Evaluates to false if the msg is not for the <replay-config> operation

Validate Phase:

The following parameters will be passed to this callback function:

- **scb** == dummy session control block for the system user (called superuser in log output)
- **msg** == dummy RPC request message for the internal <replay-config> operation
- **cbtyp** == AGT_CB_VALIDATE
- **newval** == The node from the configuration that is being replayed
- **editop** == OP_EDITOP_LOAD
- **curval** == NULL

Apply Phase:

The following parameters will be passed to this callback function:

- **scb** == dummy session control block for the system user (called superuser in log output)
- **msg** == dummy RPC request message for the internal <replay-config> operation
- **cbtyp** == AGT_CB_APPLY
- **editop** == OP_EDITOP_LOAD
- **newval** == The node from the configuration that is being replayed
- **curval** == NULL

Commit/Rollback Phase:

The following parameters will be passed to this callback function:

- **scb** == dummy session control block for the system user (called superuser in log output)

- **msg** == dummy RPC request message for the internal <replay-config> operation
- **cbtyp** == AGT_CB_COMMIT or AGT_CB_ROLLBACK
- **editop** == OP_EDITOP_LOAD
- **newval** == The node from the configuration that is being replayed
- **curval** == NULL

7.2.3 Configuration Replay Callbacks

The **agt_replay_fn_t** callback defined in **agt.h** is invoked when a configuration replay procedure is started, and then invoked again when it is finished.

The details for using this system callback are in the **YumaPro Server yp-system API Guide.**

7.2.4 Configuration Replay Cookie

There is a variant of this API that supports a user-defined cookie to be made available during the replay procedure.

Use **agt_request_replay_ex** to provide a cookie value for the configuration replay.

```
/*****************************************************************
* FUNCTION agt_request_replay_ex
*
* Request replay of the running config to SIL modules
* because SIL hardware has reset somehow
*
* Use the function agt_cfg_get_replay_cookie() to retrieve
* the cookie from a SIL callback function
*
* INPUT:
*   cookie == pointer to set as the replay_cookie.
*****************************************************************/
extern void agt_request_replay_ex (void *cookie);
```

Use the function agt_get_replay_cookie to retrieve the cookie that ws passed during the **agt_request_replay_ex()** function call. This function will return NULL if there is no replay active or if there was no cookie provided.

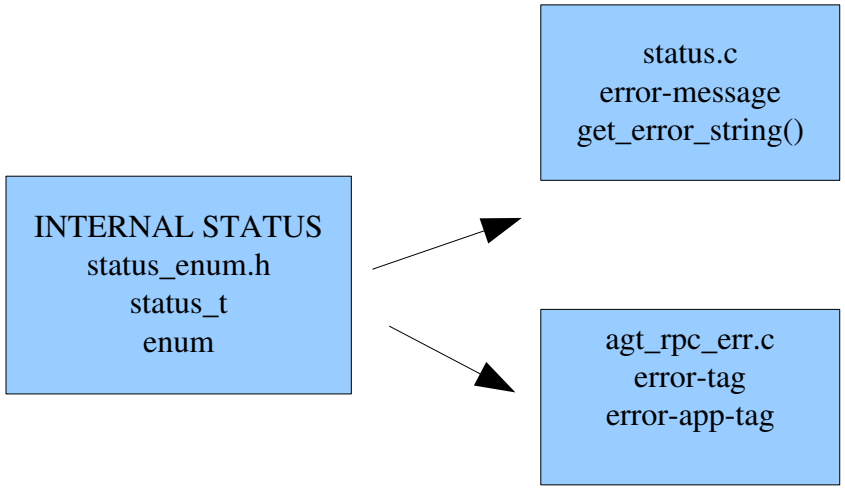
```
/*****************************************************************
* FUNCTION agt_get_replay_cookie
*
* Get the current replay cookie
*
* RETURNS:
*   cooie value
*
*****************************************************************/
extern void *
    agt_get_replay_cookie (void);
```

7.3 Error Handling

The server has several APIs for setting error information that will be returned to a client in an <rpc-error> message. Each API allows different amounts of context-specific information included in the error response.

API	Description
Return Status	The **status_t** enumeration returned by the SIL or SIL-SA code will be checked. If not error information has been recorded, then the server will construct an <rpc-error> based on the return status
agt_record_error	The **agt_record_error** API allows the basic error fields to be set
agt_record_error_errinfo	The **agt_record_error_errinfo** API allows all fields to be set, including over-riding the default error-message and error-app-tag fields.

Error Data Tables



- The internal status codes are defined in **netconf/src/ncx/status_enum.h**
- The value 0 (NO_ERR) is used to indicate success.
- Values 1 to 999 are error codes.
- Values 1000 to 1999 are warning codes
- Values 2000 to 2999 are system return codes
- Error strings are defined in **netconf/src/ncx/status.c** in the **get_error_string** function for each enumeration defined in status_enum.h

- Each **status**_t enumeration is mapped to an **error-tag** value in the file **netconf/src/agt/agt_rpcerr.c,** in the function **get_rpcerr**. This mapping also includes the default **error-app-tag** value
- The default **error-message** and **error-app-tag** values can be overridden using the YANG statements with the same name. These fields can also be specified internally (without adding these YANG statements) using the **agt_record_error_errinfo** function instead of the **agt_record_error** function.

7.3.1 <rpc-error> Contents

NETCONF has several standard error fields that are returned in an <rpc-error> response. These are defined in RFC 6241, section 4.3. They are summarized here:

Field	Description
error-type	Mandatory field set by the server indicating the protocol layer where the error occurred • transport • rpc • protocol • application
error-tag	Mandatory enumeration for the type of error that occurred. This field is set by the server depending on the status_t of the error. The values are defined in RFC 6241, Appendix A • in-use • invalid-value • too-big • missing-attribute • bad-attribute • unknown-attribute • missing-element • bad-element • unknown-element • unknown-namespace • access-denied • lock-denied • resource-denied • rollback-failed • data-exists • data-missing • operation-not-supported • operation-failed • malformed-message
error-severity	Mandatory field set by the server. The value 'error' always used because no error-tag values are defined with a severity of 'warning' • error • warning
error-app-tag	Optional field identifying the application-specific error. There are some standard values defined by YANG that must be used in certain error conditions. The server will use these values even if error-app-tag values are defined for the specific errors instead • must-violation • instance-required • missing-choice • missing-instance

Field	Description
error-path	Optional field containing an XPath absolute path expression identifying the node that caused the error. Will be present if the user input can be associated with the error.
error-message	Optional field containing a short error message. The server always generates an error-message field
error-info	This optional field is a container that has child leaf nodes representing additional error information. Some NETCONF errors require that certain error-info child nodes be present, depending on the error-tag value • bad-attribute • bad-element • bad-namespace • session-id The server also adds the folling error-info child node in every error response • error-number

7.3.2 SET_ERROR

There is a C macro called 'SET_ERROR' that is used throughout the code. This macro is only used to flag programming errors. A typical use will be in the 'default' case of a switch to flag a value that was not handled by the code.

Do not use this macro to return normal errors!

This macro will cause “assert()” to be invoked, halting program execution.

The **SET_ERROR()** macro defined in **netconf/src/ncx/status.h** is for flagging internal errors.

THIS MACRO MUST NOT BE USED FOR NORMAL ERRORS.

This macro causes some internal error and traceback information to be printed and in DEBUG builds will case an assert() exception to be generated. The following auto-generated switch statement (SIL code) has a SET_ERROR macro call for unexpected case values.

```
switch (cbtyp) {
case AGT_CB_VALIDATE:
    /* description-stmt validation here */
    break;
case AGT_CB_APPLY:
    /* database manipulation done here */
    break;
case AGT_CB_COMMIT:
    /* device instrumentation done here */
    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);
    }
    break;
case AGT_CB_ROLLBACK:
    /* undo device instrumentation here */
    break;
default:
    res = SET_ERROR(ERR_INTERNAL_VAL);
}
```

7.3.3 agt_record_error

The agt_record_error function is used to cause an error to be generated

```
/*************************************************************
* FUNCTION agt_record_error
*
* Generate an rpc_err_rec_t and save it in the msg
*
* INPUTS:
*    scb == session control block
*        == NULL and no stats will be recorded
*    msghdr == XML msg header with error Q
*          == NULL, no errors will be recorded!
*    layer == netconf layer error occured              <error-type>
*    res == internal error code                     <error-app-tag>
*    xmlnode == XML node causing error  <bad-element>   <error-path>
*            == NULL if not available
*    parmtyp == type of node in 'error_info'
*    error_info == error data, specific to 'res'        <error-info>
*               == NULL if not available (then nodetyp ignored)
*    nodetyp == type of node in 'error_path'
*    error_path == internal data node with the error       <error-path>
*            == NULL if not available or not used
* OUTPUTS:
*   errQ has error message added if no malloc errors
*   scb->stats may be updated if scb non-NULL
*
* RETURNS:
*    none
*************************************************************/
extern 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);
```

Parameter	Description
scb	Session control block for the session running at the time. Used for recording statistics in the ietf-netconf-monitoring module. May be NULL but no statistics will be recorded
msghdr	message header used to store the error information
layer	Protocol layer for the error • NCX_LAYER_TRANSPORT • NCX_LAYER_RPC • NCX_LAYER_OPERATION

	• NCX_LAYER_CONTENT
res	status_t enumeration for the error
xmlnode	XML node that caused the error, if known. Will be used for error-path information if no other information provided. May be NULL if no XML node is available (E.g. JSON or database error)
parmtyp	The parameter type enumeration identifying what type of data is passed in the error_info parameter. These 2 fields are required if the specific error-tag requires error-info data to be included. • NCX_NT_NONE (error_info == NULL) • NCX_NT_STRING (error_info == const xmlChar *) • NCX_NT_OBJ (error_info == const obj_template_t *) • NCX_NT_VAL (error_info == const val_value_t *) The following types are used for special errors: • NCX_NT_QNAME (error_info == const xmlns_qname_t *) • NCX_NT_CFG (error_info == const cfg_template_t *) • NCX_NT_UINT32_PTR (error_info = const uint32 *)
error_info	This void pointer is cast to different types based on the value of the parmtyp parameter. It is used for the <error-info> child node in the <rpc-error> response message
nodetyp	The parameter type enumeration identifying what type of data is passed in the error_path parameter. These 2 fields are optional and are used to fill in the error-path field in the <rpc-error> message • NCX_NT_NONE (error_path == NULL) • NCX_NT_STRING (error_path == const xmlChar *) • NCX_NT_OBJ (error_path == const obj_template_t *) • NCX_NT_VAL (error_path == const val_value_t *)
error_path	This void pointer is cast to different types based on the value of the nodetyp parameter. It is used for the <error-path> child node in the <rpc-error> response message

7.3.4 agt_record_error_errinfo

The **agt_record_error_errinfo** function is the same as the **agt_record_error** function, except it also allows the 'errinfo' parameter to be specified. The **ncx_errinfo_t** struct in **netconf/src/ncx/ncxtypes.h** is used to contain this error information.

Parameter	Description
errinfo	Pointer to a structure containing some of the error information to use. • **error_app_tag:** string for <error-app-tag> field • **error_message**: string for <error-message> field

```
/********************************************************************
* FUNCTION agt_record_error_errinfo
*
* Generate an rpc_err_rec_t and save it in the msg
* Use the provided error info record for <rpc-error> fields
*
* INPUTS:
*    scb == session control block
*        == NULL and no stats will be recorded
*    msghdr == XML msg header with error Q
*          == NULL, no errors will be recorded!
*    layer == netconf layer error occured              <error-type>
*    res == internal error code                      <error-app-tag>
*    xmlnode == XML node causing error  <bad-element>   <error-path>
*            == NULL if not available
*    parmtyp == type of node in 'error_info'
*    error_info == error data, specific to 'res'        <error-info>
*               == NULL if not available (then nodetyp ignored)
*    nodetyp == type of node in 'error_path'
*    error_path == internal data node with the error       <error-path>
*            == NULL if not available or not used
*    errinfo == error info record to use
*
* OUTPUTS:
*   errQ has error message added if no malloc errors
*   scb->stats may be updated if scb non-NULL

* RETURNS:
*    none
*********************************************************************/
extern 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_parm,
                          ncx_node_t nodetyp,
                          void *error_path,
                          const ncx_errinfo_t *errinfo);
```

7.3.5 agt_record_warning

The NETCONF protocol does not support warnings. Use of this API will not be compatible with the NETCONF standard.

If the **--with-warnings** CLI parameter is set to “true” then the server will allow the error-severity field to be set to “warning”.

The agt_record_warning API function can be used to set the error-severity field to “warning”.

If the **--with-warnings** parameter is set to FALSE (the default) then the agt_record_warning function will set the error-severity field to “error” instead of “warning”. This function has the same parameters as **agt_record_error_errinfo**.

```
/*************************************************************
* FUNCTION agt_record_warning
*
* Generate an rpc_err_rec_t and save it in the msg
* Use the provided error info record for <rpc-error> fields
* Set the error-severity field to warning instead of error
* but only if agt_with_warnings is TRUE
*
* INPUTS:
*    scb == session control block
*        == NULL and no stats will be recorded
*    msghdr == XML msg header with error Q
*          == NULL, no errors will be recorded!
*    layer == netconf layer error occured              <error-type>
*    res == internal error code                      <error-app-tag>
*    xmlnode == XML node causing error  <bad-element>   <error-path>
*            == NULL if not available
*    parmtyp == type of node in 'error_info'
*    error_info == error data, specific to 'res'        <error-info>
*               == NULL if not available (then nodetyp ignored)
*    nodetyp == type of node in 'error_path'
*    error_path == internal data node with the error       <error-path>
*            == NULL if not available or not used
*    errinfo == error info record to use (may be NULL if not used)
*
* OUTPUTS:
*   errQ has error message added if no malloc errors
*   scb->stats may be updated if scb non-NULL

* RETURNS:
*    none
*************************************************************/
extern void
    agt_record_warning (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_parm,
                        ncx_node_t nodetyp,
                        void *error_path,
                        const ncx_errinfo_t *errinfo);
```

7.3.6 Adding <error-info> Data to an Error Response

The NETCONF <rpc-error> and YANG-API/RESTCONF <errors> data structures allow error data to be reported to the client. This data can be located in a container called “error-info”. Typically, the server adds an “error-number” leaf to this container, but there are several NETCONF errors that require specific error data to be returned.

Step 1) Define the error-info data

The new API functions require the following H files, which need to be included into the SIL C file:

```
#include “agt_util.h”
#include “rpc.h”
```

The data returned to an <error-info> record should be defined in a real YANG module.

In the example below, an abstract leaf called 'testdata' is defined:

```
  leaf testdata {
    ncx:abstract;
    type int32;
    description
      "Example of template for user <error-info> data";
  }
```

Step 2) Create an instance of the data to add to the error response.

This must be malloced so it cal be freed with the **val_free_value()** function

The example from **sil_error/src/sil-error.c** shows an instance of testdata being created:

```
/********************************************************************
* FUNCTION add_user_error_data
*
* Add an example error-info data node to the error that is returned
* to the client
*
* INPUTS:
*    msg == RPC message to add user error data
* RETURNS:
*   status
********************************************************************/
static status_t
    add_user_error_data (rpc_msg_t *msg)
{
    /* get the abstract leaf template */
    obj_template_t *testdata_obj =
```

```
        obj_find_template_top(sil_error_mod,
                              sil_error_mod->name,
                              (const xmlChar *)"testdata");

    if (testdata_obj == NULL) {
        return ERR_NCX_DEF_NOT_FOUND;
    }

    /* 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);
    }

    return res;

}  /* add_user_error_data */
```

Step 3) Add the new data value to the message error data queue.

This is done with the **rpc_msg_add_error_data()** API function. This function can be called multiple times, to add multiple data nodes to the error response.

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

This step is done BEFORE the agt_record_error (or other variant) is called by the SIL code:

```
    if (res != NO_ERR) {

        /* add user data to the message before calling agt_record_error */
        (void)add_user_error_data(msg);

        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_CONTENT,
            res,
            NULL,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval);
    }
```

Example Error Message

The “sil-trigger” object in the previous example is used to show how error-info data is added to the error response.

The error is triggered by setting **/sil-phase** to 'apply' and **/sil-trigger** to any value.

> merge /sil-trigger value=2

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="4" trace-id="4"
 xmlns:silerr="http://www.netconfcentral.org/ns/sil-error"
 xmlns:ncx="http://netconfcentral.org/ns/yuma-ncx"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
  <error-tag>operation-failed</error-tag>
  <error-severity>error</error-severity>
  <error-app-tag>general-error</error-app-tag>
  <error-path>/silerr:sil-trigger</error-path>
  <error-message xml:lang="en">operation failed</error-message>
  <error-info>
   <error-number>274</error-number>
   <testdata xmlns="http://www.netconfcentral.org/ns/sil-error">42</testdata>
  </error-info>
 </rpc-error>
</rpc-reply>
```

The following example shows how YANG-API will return the user error data:

POST /yang-api/datastore/sil-trigger HTTP/1.1

```
{
 "errors": {
  "error": [
   {
    "error-type":"application",
    "error-tag":"operation-failed",
    "error-app-tag":"general-error",
    "error-path":"/silerr:sil-trigger",
    "error-message":"operation failed",
    "error-info": {
     "error-number":274,
     "testdata":42
    }
   }
  ]
 }
}
```

The following example shows how RESTCONF will return the user error data:

If any errors occur while attempting to invoke the operation, then an "errors" data structure is returned with the appropriate error status.

The client might send the following POST request message:

POST /restconf/operations/example-ops:reboot HTTP/1.1

Accept: application/yang-data+xml

Content-Type: application/yang-data+xml

```
<input xmlns="https://example.com/ns/example-ops">
 <delay>-33</delay>
 <message>Going down for system maintenance</message>
 <language>en-US</language>
</input>
```

The server might respond with an "invalid-value" error in default XML foramt:

HTTP/1.1 400 Bad Request

Date: Mon, 25 Apr 2012 11:10:30 GMT

Server: example-server

Content-Type: application/yang-data+xml

```
<errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
 <error>
  <error-type>protocol</error-type>
  <error-tag>invalid-value</error-tag>
  <error-path xmlns:err="https://example.com/ns/example-ops">
    err:input/err:delay
  </error-path>
  <error-message>Invalid input parameter</error-message>
 </error>
</errors>
```

Or, if user specifies application/yang-data+json Accept header entry, the server might respond:

HTTP/1.1 409 Conflict

Date: Mon, 23 Apr 2012 17:11:00 GMT

Server: example-server

Content-Type: application/yang-data+json

```
{
  "ietf-restconf:errors": {
    "error": [
      {
        "error-type": "protocol",
        "error-tag": "lock-denied",
        "error-message": "Lock failed, lock already held"
```

```
            }
          ]
        }
      }
```

7.4 RPC Operation Interface

All RPC operations are data-driven within the server, using the YANG rpc statement for the operation and SIL callback functions.

Any new protocol operation can be added by defining a new YANG rpc statement in a module, and providing the proper SIL code.

7.4.1 RPC Callback Template

The **agt_rpc_method_t** function in agt/agt_rpc.h is used as the callback template for all RPC callback phases.

```
/* Template for RPC server callbacks
 * The same template is used for all RPC callback phases
 */
typedef status_t
    (*agt_rpc_method_t) (ses_cb_t *scb,
                         rpc_msg_t *msg,
                         xml_node_t *methnode);
```

agt_rpc_method_t

Parameter	Description
scb	The session control block for the session handling RPC request
msg	The RPC message in progress containing the input parameters (if any)
methnode	The XML node being parsed if this is available. This can used for error reporting if no 'val' or 'obj' parameters are available (from the request message)

7.4.2 RPC Callback Initialization

The **agt_rpc_register_method** function in **agt/agt_rpc.h** is used to provide a callback function for a specific callback phase. The same function can be used for multiple phases if desired.

```
extern status_t
    agt_rpc_register_method (const xmlChar *module,
                             const xmlChar *method_name,
                             agt_rpc_phase_t  phase,
                             agt_rpc_method_t method);
```

agt_rpc_register_method

Parameter	Description
module	The name of the module that contains the rpc statement
method_name	The identifier for the rpc statement
phase	AGT_PH_VALIDATE(0): validate phase AGT_PH_INVOKE(1): invoke phase AGT_PH_POST_REPLY(2): post-reply phase
method	The address of the callback function to register

7.4.3 RPC Message Header

The NETCONF server will parse the incoming XML message and construct an RPC message header, which is used to maintain state and any other message-specific data during the processing of an incoming <rpc> request.

The following C code represents the **rpc_msg_t** data structure:

```
/* NETCONF Server and Client RPC Request/Reply Message Header */
typedef struct rpc_msg_t_ {
    dlq_hdr_t        qhdr;

    /* generic XML message header */
    xml_msg_hdr_t    mhdr;

    /* incoming: top-level rpc element data */
    xml_attrs_t     *rpc_in_attrs;     /* borrowed from <rpc> elem */

    /* incoming:
     * 2nd-level method name element data, used in agt_output_filter
     * to check get or get-config
     */
    struct obj_template_t_ *rpc_method;

    /* incoming: SERVER RPC processing state */
    int              rpc_agt_state;        /* agt_rpc_phase_t */
    op_errop_t       rpc_err_option;
    op_editop_t      rpc_top_editop;
    val_value_t     *rpc_input;

    /* incoming:
     * hooks for method routines to save context or whatever
     */
    void           *rpc_user1;
    void           *rpc_user2;
    uint32          rpc_returncode;   /* for nested callbacks */

    /* incoming: get method reply handling builtin
     * If the rpc_datacb is non-NULL then it will be used as a
     * callback to generate the rpc-reply inline, instead of
     * buffering the output.
     * The rpc_data and rpc_filter parameters are optionally used
     * by the rpc_datacb function to generate a reply.
     */
    rpc_data_t      rpc_data_type;          /* type of data reply */
     void           *rpc_datacb;             /* agt_rpc_data_cb_t */
    dlq_hdr_t       rpc_dataQ;       /* data reply: Q of val_value_t */
    op_filter_t     rpc_filter;        /* backptrs for get* methods */

    /* incoming: rollback or commit phase support builtin
     * As an edit-config (or other RPC) is processed, a
     * queue of 'undo records' is collected.
     * If the apply phase succeeds then it is discarded,
     * otherwise if rollback-on-error is requested,
     * it is used to undo each change that did succeed (if any)
     * before an error occured in the apply phase.
     */
    boolean          rpc_need_undo;   /* set by edit_config_validate */
    dlq_hdr_t        rpc_undoQ;       /* Q of rpc_undo_rec_t */
    dlq_hdr_t        rpc_auditQ;       /* Q of rpc_audit_rec_t */
```

```
} rpc_msg_t;
```

The **rpc_msg_t** data structure in **ncx/rpc.h** is used for this purpose. The following table summarizes the fields:

rpc_msg_t

Field	Type	User Mode	Description
qhdr	dlq_hdr_t	none	Queue header to store RPC messages in a queue (within the session header)
mhdr	xml_msg_hdr_t	none	XML message prefix map and other data used to parse the request and generate the reply.
rpc_in_attrs	xml_attrs_t *	read write	Queue of xml_attr_t representing any XML attributes that were present in the <rpc> element. A callback function may add xml_attr_t structs to this queue to send in the reply.
rpc_method	obj_template_t *	read	Back-pointer to the object template for this RPC operation.
rpc_agt_state	int	read	Enum value (0, 1, 2) for the current RPC callback phase.
rpc_err_option	op_errop_t	read	Enum value for the <error-option> parameter. This is only set if the <edit-config> operation is in progress.
rpc_top_editop	op_editop_t	read	Enum value for the <default-operation> parameter. This is only set if the <edit-config> operation is in progress.
rpc_input	val_value_t *	read	Value tree representing the container of 'input' parameters for this RPC operation.
rpc_user1	void *	read write	Void pointer that can be used by the SIL functions to store their own message-specific data.
rpc_user2	void *	read write	Void pointer that can be used by the SIL functions to store their own message-specific data.
rpc_returncode	uint32	none	Internal return code used to control nested callbacks.
rpc_data_type	rpc_data_t	write	For RPC operations that return data, this enumeration is set to indicate which type of data is desired. **RPC_DATA_STD**: A <data> container will be used to encapsulate any returned data, within the <rpc-reply> element. **RPC_DATA_YANG**: The <rpc-reply> element will be the only container encapsulated any returned data.
rpc_datacb	void *	write	For operations that return streamed data, this pointer is set to the desired callback function to use for generated the data portion of the <rpc-

Field	Type	User Mode	Description
			reply> XML response. The template for this callback is **agt_rpc_data_cb_t,** found in agt_rpc.h
rpc_dataQ	dlq_hdr_t	write	For operations that return stored data, this queue of val_value_t structures can be used to provide the response data. Each val_value_t structure will be encoded as one of the corresponding RPC output parameters.
rpc_filter	op_filter_t	none	Internal structure for optimizing subtree and XPath retrieval operations.
rpc_need_undo	boolean	none	Internal flag to indicate if rollback-on-error is in effect dusing an <edit-config> operation.
rpc_undoQ	dlq_hdr_t	none	Queue of **rpc_undo_rec_t** structures, used to undo edits if rollback-on-error is in affect during an <edit-config> operation.
rpc_auditQ	dlq_hdr_t	none	Queue of rpc_audit_rec_t structures used internally to generate database alteration notifications and audit log entries.

7.4.4 SIL Support Functions For RPC Operations

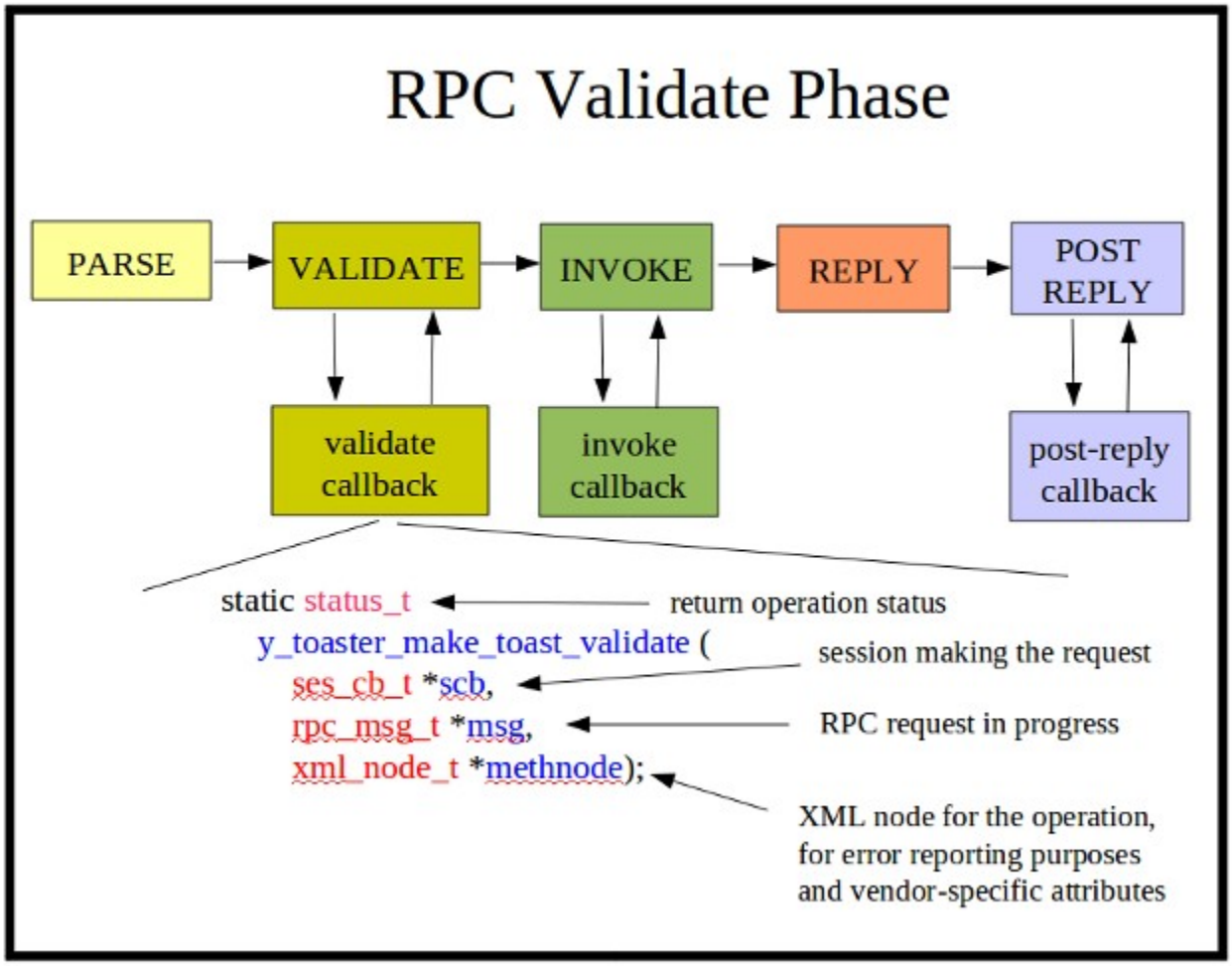
The file **agt/agt_rpc.c** contains some functions that are used by SIL callback functions.

The following table highlights the functions that may be useful to SIL developers:

agt/agt_rpc.c Functions

Function	Description
agt_rpc_register_method	Register a SIL RPC operation callback function for 1 callback phase.
agt_rpc_support_method	Tell the server that an RPC operation is supported by the system..
agt_rpc_unsupport_method	Tell the server that an RPC operation is not supported by the system..
agt_rpc_unregister_method	Remove all the SIL RPC operation callback functions for 1 RPC operation.

7.4.5 RPC Validate Callback Function



The RPC validate callback function is optional to use. Its purpose is to validate any aspects of an RPC operation, beyond the constraints checked by the server engine. Only 1 validate function can register for each YANG rpc statement. The standard NETCONF operations are reserved by the server engine. There is usually zero or one of these callback functions for every 'rpc' statement in the YANG module associated with the SIL code.

It is enabled with the **agt_rpc_register_method** function, within the phase 1 initialization callback function.

The **yangdump-sdk** code generator will create this SIL callback function by default. There will be C comments in the code to indicate where your additional C code should be added.

The **val_find_child** function is commonly used to find particular parameters within the RPC input section, which is encoded as a **val_value_t** tree.

The **agt_record_error** function is commonly used to record any parameter or other errors. In the **libtoaster** example, there are internal state variables (**toaster_enabled** and **toaster_toasting**), maintained by the SIL code, which are checked in addition to any provided parameters.

Example RPC YANG definition:

```
rpc make-toast {
    description
      "Make some toast.
       The toastDone notification will be sent when
       the toast is finished.
       An 'in-use' error will be returned if toast
       is already being made.
       A 'resource-denied' error will be returned
       if the toaster service is disabled.";
    input {
        leaf toasterDoneness {
            type uint32 {
                range "1 .. 10";
            }
            default 5;
            description
              "This variable controls how well-done is the
               ensuing toast. It should be on a scale of 1 to 10.
               Toast made at 10 generally is considered unfit
               for human consumption; toast made at 1 is warmed
               lightly.";
        }
        leaf toasterToastType {
            type identityref {
                base toast:toast-type;
            }
            default toast:wheat-bread;
            description
              "This variable informs the toaster of the type of
               material that is being toasted. The toaster
               uses this information, combined with
               toasterDoneness, to compute for how
               long the material must be toasted to achieve
               the required doneness.";
        }
    }
}
```

Example SIL Function Registration:

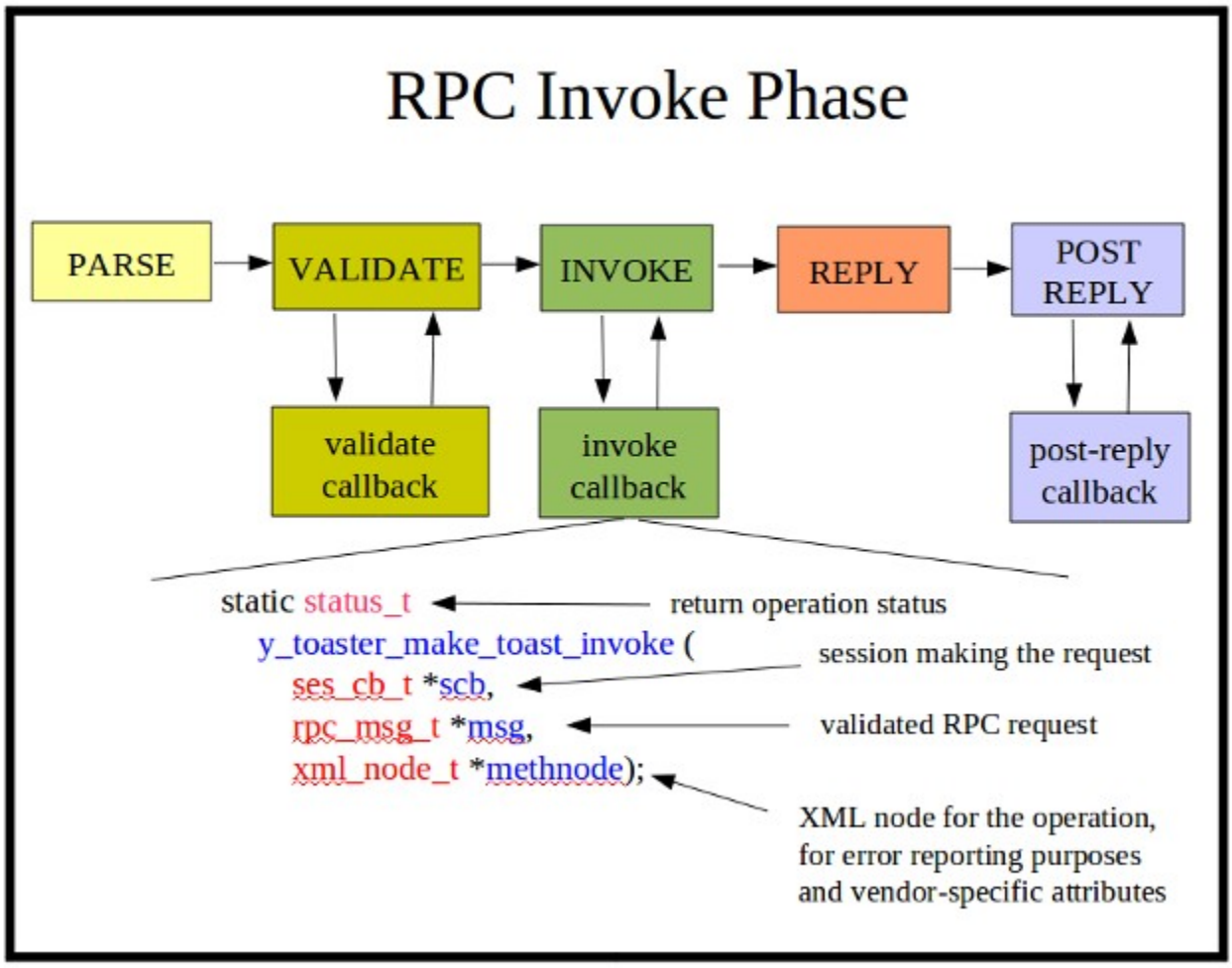
```
res = agt_rpc_register_method(
    y_toaster_M_toaster,
    y_toaster_N_make_toast,
    AGT_RPC_PH_VALIDATE,
    y_toaster_make_toast_validate);
if (res != NO_ERR) {
    return res;
}
```

Example SIL Function:

```
/********************************************************************
* FUNCTION y_toaster_make_toast_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_toaster_make_toast_validate (
        ses_cb_t *scb,
        rpc_msg_t *msg,
        xml_node_t *methnode)
{
    status_t res;
    val_value_t *errorval;
    const xmlChar *errorstr;
    val_value_t *toasterDoneness_val;
    val_value_t *toasterToastType_val;
    uint32 toasterDoneness;
    val_idref_t *toasterToastType;

    res = NO_ERR;
    errorval = NULL;
    errorstr = NULL;

    toasterDoneness_val = val_find_child(
        msg->rpc_input,
        y_toaster_M_toaster,
        y_toaster_N_toasterDoneness);
    if (toasterDoneness_val != NULL && toasterDoneness_val->res == NO_ERR) {
        toasterDoneness = VAL_UINT(toasterDoneness_val);
    }

    toasterToastType_val = val_find_child(
        msg->rpc_input,
        y_toaster_M_toaster,
        y_toaster_N_toasterToastType);
    if (toasterToastType_val != NULL && toasterToastType_val->res == NO_ERR) {
        toasterToastType = VAL_IDREF(toasterToastType_val);
    }



    /* added code starts here */
    if (toaster_enabled) {
        /* toaster service enabled, check if in use */


        if (toaster_toasting) {
            res = ERR_NCX_IN_USE;
        } else {
            /* this is where a check on bread inventory would go */

            /* this is where a check on toaster HW ready would go */
```

```
        }
    } else {
        /* toaster service disabled */
        res = ERR_NCX_RESOURCE_DENIED;
    }
    /* added code ends here */

    /* if error: set the res, errorstr, and errorval parms */
    if (res != NO_ERR) {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_OPERATION,
            res,
            methnode,
            NCX_NT_STRING,
            errorstr,
            NCX_NT_VAL,
            errorval);
    }

    return res;

} /* y_toaster_make_toast_validate */
```

7.4.6 RPC Invoke Callback Function



The RPC invoke callback function is used to perform the operation requested by the client session. Only 1 invoke function can register for each YANG rpc statement. The standard NETCONF operations are reserved by the server engine. There is usually one of these callback functions for every 'rpc' statement in the YANG module associated with the SIL code.

The RPC invoke callback function is optional to use, although if no invoke callback is provided, then the operation will have no affect. Normally, this is only the case if the module is be tested by an application developer, using **netconfd-pro** as a server simulator.

It is enabled with the **agt_rpc_register_method** function, within the phase 1 initialization callback function.

The **yangdump-sdk** code generator will create this SIL callback function by default. There will be C comments in the code to indicate where your additional C code should be added.

The **val_find_child** function is commonly used to retrieve particular parameters within the RPC input section, which is encoded as a **val_value_t** tree. The **rpc_user1** and **rpc_user2** cache pointers in the **rpc_msg_t** structure can also be used to store data in the validation phase, so it can be immediately available in the invoke phase.

The **agt_record_error** function is commonly used to record any internal or platform-specific errors. In the **libtoaster** example, if the request to create a timer callback control block fails, then an error is recorded.

For RPC operations that return either an <ok> or <rpc-error> response, there is nothing more required of the RPC invoke callback function.

For operations which return some data or <rpc-error>, the SIL code must do 1 of 2 additional tasks:

- add a **val_value_t** structure to the **rpc_dataQ** queue in the **rpc_msg_t** for each parameter listed in the YANG rpc 'output' section.
- set the rpc_datacb pointer in the rpc_msg_t structure to the address of your data reply callback function. See the **agt_rpc_data_cb_t** definition in **agt/agt_rpc.h** for more details.

Example SIL Function Registration

```
    res = agt_rpc_register_method(
        y_toaster_M_toaster,
        y_toaster_N_make_toast,
        AGT_RPC_PH_INVOKE,
        y_toaster_make_toast_invoke);
    if (res != NO_ERR) {
        return res;
    }
```

Example SIL Function:

```
/********************************************************************
* FUNCTION y_toaster_make_toast_invoke
*
* RPC invocation phase
* All constraints have passed at this point.
* Call device instrumentation code in this function.
*
* INPUTS:
*     see agt/agt_rpc.h for details
*
* RETURNS:
*     error status
********************************************************************/
static status_t
    y_toaster_make_toast_invoke (
        ses_cb_t *scb,
        rpc_msg_t *msg,
        xml_node_t *methnode)
{
    status_t res;
    val_value_t *toasterDoneness_val;
    val_value_t *toasterToastType_val;
    uint32 toasterDoneness;
    val_idref_t *toasterToastType;

    res = NO_ERR;
    toasterDoneness = 0;

    toasterDoneness_val = val_find_child(
        msg->rpc_input,
        y_toaster_M_toaster,
        y_toaster_N_toasterDoneness);
    if (toasterDoneness_val != NULL && toasterDoneness_val->res == NO_ERR) {
        toasterDoneness = VAL_UINT(toasterDoneness_val);
    }
```

```
    toasterToastType_val = val_find_child(
        msg->rpc_input,
        y_toaster_M_toaster,
        y_toaster_N_toasterToastType);
    if (toasterToastType_val != NULL && toasterToastType_val->res == NO_ERR) {
        toasterToastType = VAL_IDREF(toasterToastType_val);
    }

    /* invoke your device instrumentation code here */

    /* make sure the toasterDoneness value is set */
    if (toasterDoneness_val == NULL) {
        toasterDoneness = 5;   /* set the default */
    }


    /* arbitrary formula to convert toaster doneness to the
     * number of seconds the toaster should be on
     */
    toaster_duration = toasterDoneness * 12;

    /* this is where the code would go to adjust the duration
     * based on the bread type
     */

    if (LOGDEBUG) {
        log_debug("\ntoaster: starting toaster for %u seconds",
                  toaster_duration);
    }

    /* this is where the code would go to start the toaster
     * heater element
     */

    /* start a timer to toast for the specified time interval */
    res = agt_timer_create(toaster_duration,
                           FALSE,
                           toaster_timer_fn,
                           NULL,
                           &toaster_timer_id);
    if (res == NO_ERR) {
        toaster_toasting = TRUE;
    } else {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_OPERATION,
            res,
            methnode,
            NCX_NT_NONE,
            NULL,
            NCX_NT_NONE,
            NULL);
    }
    /* added code ends here */

    return res;

} /* y_toaster_make_toast_invoke */
```

7.4.7 RPC Data Output Handling

RPC operations can return data to the client if the operation succeeds. The YANG “output” statement defines the return data for each RPC operation. Constructing YANG data is covered in detail elsewhere. This section shows a simple example SIL invoke function that returns data.

The output data is usually constructed with APIs like agt_make_leaf from **agt_util.h**.

To use these APIs, the output object is needed.

The following code can be used to get this object:

```
   obj_template_t *obj = RPC_MSG_METHOD(msg);
   if (obj) {
       obj = obj_find_child(obj, NULL, NCX_EL_OUTPUT);
   }
   if (obj == NULL) {
       return ERR_NCX_DEF_NOT_FOUND;  // should not happen
   }
```

Once this object template is retrieved from the **msg** parameter data can be added to the **msg** using **val_value_t** structures.

The following snippet shows a leaf being created using an **int32** value.

```
  val_value_t *val =
       agt_make_int_leaf(obj,
                         y_addrpc_N_sum,
                         result,
                         &res);
```

After the value is created it must be added to the message using the **agt_rpc_add_return_val** API:

```
   if (val) {
       agt_rpc_add_return_val(val, msg);
   }
```

Example YANG Module:

```
module addrpc {
  namespace "http://www.yumaworks.com/ns/addrpc";
  prefix add;
  revision "2020-02-25";

  rpc add {
```

```
    description "Get the sum of two numbers";
    input {
      leaf num1 {
        type int32;
        mandatory true;
        description "First number to add";
      }
      leaf num2 {
        type int32;
        mandatory true;
        description "Second number to add";
      }
    }
    output {
      leaf sum {
        type int32;
        mandatory true;
        description "The sum of the 2 numbers";
      }
    }
   }
}
```

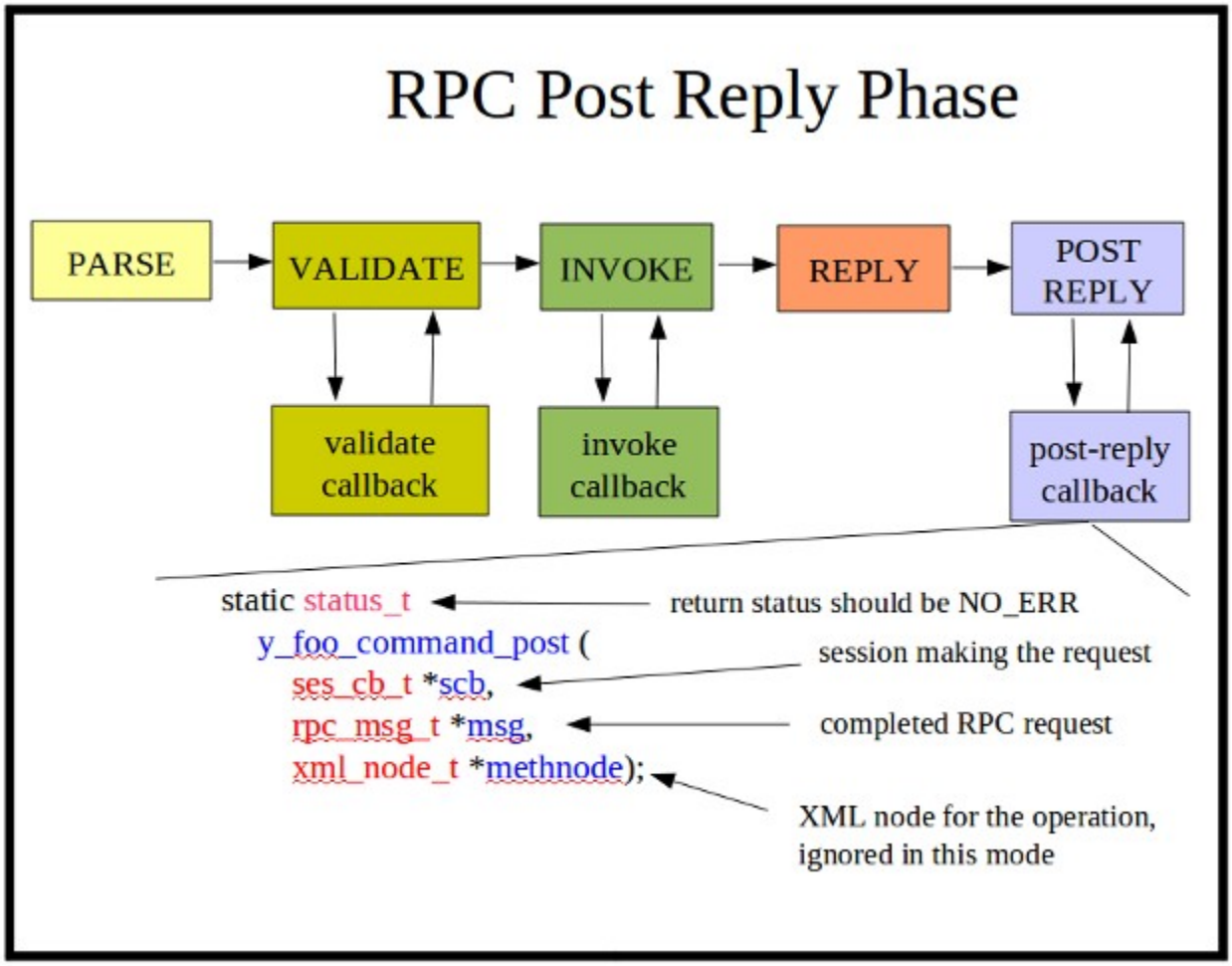
Example SIL Invoke Function Returning Output Data:

(The **bold** text added after SIL stub generated).

```
/********************************************************************
* FUNCTION y_addrpc_add_invoke
*
* RPC invocation phase
* All constraints have passed at this point.
* Call device instrumentation code in this function.
*
* INPUTS:
*     see agt/agt_rpc.h for details
*
* RETURNS:
*     error status
********************************************************************/
static status_t y_addrpc_add_invoke (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    xml_node_t *methnode)
{
    status_t res = NO_ERR;

    if (LOGDEBUG) {
        log_debug("\nStart SIL invoke rpc <add> from module addrpc");
    }

    val_value_t *inputval = agt_get_rpc_input(msg);
    if (inputval == NULL) return ERR_NCX_OPERATION_FAILED;

    val_value_t *v_num1_val = NULL;
    int32 v_num1 = 0;

    val_value_t *v_num2_val = NULL;
```

```
    int32 v_num2 = 0;

    v_num1_val = val_find_child(
        inputval,
        y_addrpc_M_addrpc,
        y_addrpc_N_num1);
    if (v_num1_val) {
        v_num1 = VAL_INT(v_num1_val);
    }

    v_num2_val = val_find_child(
        inputval,
        y_addrpc_M_addrpc,
        y_addrpc_N_num2);
    if (v_num2_val) {
        v_num2 = VAL_INT(v_num2_val);
    }

    /* remove the next line if scb is used */
    (void)scb;

    /* remove the next line if methnode is used */
    (void)methnode;

    /* invoke your device instrumentation code here */

    /* Following output nodes expected:
     * leaf sum
     */
    int32 result = v_num1 + v_num2;
    obj_template_t *obj = RPC_MSG_METHOD(msg);
    if (obj) {
        obj = obj_find_child(obj, NULL, NCX_EL_OUTPUT);
    }
    if (obj == NULL) {
        return ERR_NCX_DEF_NOT_FOUND;  // should not happen
    }

    val_value_t *val =
        agt_make_int_leaf(obj,
                          y_addrpc_N_sum,
                          result,
                          &res);
    if (val) {
        agt_rpc_add_return_val(val, msg);
    }

    return res;

} /* y_addrpc_add_invoke */
```

7.4.8 RPC Post Reply Callback Function



The RPC post-reply callback function is used to clean up after a message has been processed. Only 1 function can register for each YANG rpc statement. The standard NETCONF operations are reserved by the server engine. This callback is not needed unless the SIL validate or invoke callback allocated some memory that needs to deleted after the <rpc-reply> is sent.

The RPC post reply callback function is optional to use. It is enabled with the **agt_rpc_register_method** function, within the phase 1 initialization callback function.

The **yangdump-pro** code generator will not create this SIL callback function by default.

Example SIL Function Registration

```
res = agt_rpc_register_method(
    y_foo_M_foo,
    y_foo_N_command,
    AGT_RPC_PH_POST_REPLY,
    y_foo_command_post);
if (res != NO_ERR) {
```

```
        return res;
    }
```

Example SIL Function:

```
/********************************************************************
* FUNCTION y_foo_command_post
*
* RPC post reply phase
*
* INPUTS:
*     see agt/agt_rpc.h for details
*
* RETURNS:
*     error status
********************************************************************/
static status_t
    y_foo_command_post (
        ses_cb_t *scb,
        rpc_msg_t *msg,
        xml_node_t *methnode)
{
    (void)scb;
    (void)methnode;
    if (msg->rpc_user1 != NULL) {
        m__free(msg->rpc_user1);
        msg->rpc_user1 = NULL;
    }
    return NO_ERR;
}  /* y_foo_command_post */
```

7.5 YANG 1.1 Action Interface

A YANG action is like an RPC operation in many ways except it is associated with the data node where the action is defined. The SIL code for an action callback is a combination of the RPC callbacks and the EDIT callbacks. The call flow is the same as for RPC operations. The parameters are similar as well, except the ancestor keys for the instance are provided, so the SIL callback knows which instance to apply the action.

All YANG actions are data-driven within the server, using the YANG action statement for the operation and SIL callback functions.

Any new data-specific actions can be added by defining a new YANG action statement within a container or list, and providing the proper SIL code.

When using the make_sil_dir_pro script for YANG actions, the --split parameter should always be used. This allows the keys to be easily processed. The examples shown below are done using the following command:

```
> make_sil_dir_pro --split ex-action
```

Example YANG Action

```
module ex-action {
  yang-version 1.1;
  namespace "http://netconfcentral.org/ns/ex-action";
  prefix exa;
  import ietf-yang-types { prefix yang; }
  revision 2020-03-06;

  list server {
    key name;
    leaf name {
      type string;
      description "Server name";
    }
    action reset {
      input {
        leaf reset-msg {
          type string;
          description "Log message to print before server reset";
        }
      }
      output {
        leaf reset-finished-at {
          type yang:date-and-time;
          description "Time the reset was done on the server";
        }
      }
    }
  }
}
```

Example YANG Action Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="1"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <action xmlns="urn:ietf:params:xml:ns:yang:1">
   <server xmlns="http://netconfcentral.org/ns/ex-action">
     <name>test1</name>
     <reset>
       <reset-msg>diagnostic mode 1</reset-msg>
     </reset>
   </server>
 </action>
</rpc>
```

Example YANG Action Reply:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="1"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <reset-finished-at xmlns="http://netconfcentral.org/ns/ex-action">2020-03-
08T19:01:22Z</reset-finished-at>
</rpc-reply>
```

7.5.1 Action Callback Template

The **agt_action_register_action** function in **agt/agt_action.h** is used to provide a callback function for a specific callback phase. The same function can be used for multiple phases if desired.

```
/* Template for Action server callbacks
 * The same template is used for all Action callback phases
 * The callback is expected to validate if needed and then
 * invoke if needed.
 *
 * The entire input hierarchy for an action is contained in
 * the msg->rpc_input node.  The 'actionval' node passed
 * to the callback points at the action node nested in the
 * input (like <ping> in the example below)

 * INPUTS:
 *   scb == session invoking the RPC
 *   msg == message in progress for this <rpc> request
 *   methnode == XML node for the operation, which can be used
 *               in error reporting (or ignored
 *   actionval == the nested 'action-method-name' node that was
 *             parsed within the topval subtree, in the
 *             RPC <action> request; this is used to help derive
 *             the list keys
 * RETURNS:
 *   status == return status for the phase; an error in validate phase
 *             will cancel invoke phase; an rpc-error will be added
 *             if an error is returned and the msg error Q is empty
 */
typedef status_t
    (*agt_action_cb_t) (ses_cb_t *scb,
                        rpc_msg_t *msg,
                        xml_node_t *methnode,
                        val_value_t *actionval);
```

agt_action_cb_t

Parameter	Description
scb	The session control block for the session handling action request
msg	The RPC message in progress containing the input parameters (if any)
methnode	The XML node being parsed if this is available. This can used for error reporting if no 'val' or 'obj' parameters are available (from the request message)
actionval	The input node represents the start of the specific action being invoked. The auto-generated SIL code can derive all the ancestor keys from this data structure.

7.5.2 Action Callback Initialization

agt_action_register_action

Parameter	Description
module	The name of the module that contains the rpc statement
method_name	The identifier for the rpc statement
phase	AGT_PH_VALIDATE(0): validate phase AGT_PH_INVOKE(1): invoke phase AGT_PH_POST_REPLY(2): post-reply phase
method	The address of the callback function to register

7.5.3 Action Message Header

The NETCONF server will parse the incoming XML message and construct an RPC message header, which is used to maintain state and any other message-specific data during the processing of an incoming <rpc> request.

The **rpc_msg_t** data structure in **ncx/rpc.h** is used for this purpose. This is exactly the same as for an RPC operation.

7.5.4 SIL Support Functions For YANG Actions

The file **agt/agt_action.c** contains some functions that are used by SIL callback functions.

agt/agt_action.c Functions

Function	Description
agt_action_register_action	Register a SIL action callback function for 1 callback phase.
agt_action_unregister_action	Remove all the SIL action callback functions for 1 action.

7.5.5 Action Validate Callback Function

The action validate callback function is optional to use. Its purpose is to validate any aspects of an cation request, beyond the constraints checked by the server engine. Only 1 validate function can register for each YANG action statement. There is usually zero or one of these callback functions for every 'action' statement in the YANG module associated with the SIL code.

It is enabled with the **agt_action_register_action** function, within the phase 1 initialization callback function.

The **yangdump-sdk** code generator will create this SIL callback function by default. There will C comments in the code to indicate where your additional C code should be added.

The **val_find_child** function is commonly used to find particular parameters within the RPC input section, which is encoded as a **val_value_t** tree.

The **agt_record_error** function is commonly used to record any parameter or other errors.

Example SIL Function Registration (found in **y_ex-action.c**)

```
    res = agt_action_register_action(
        (const xmlChar *)"/exa:server/exa:reset",
        AGT_RPC_PH_VALIDATE,
        ex_action_server_reset_action_val);
    if (res != NO_ERR) {
        return res;
    }
```

Example User SIL Function: (Note that the input parameters include the list key 'k_server_name' for the YANG list /server)

(The **bold** text added after SIL stub generated).

```
/********************************************************************
* FUNCTION u_ex_action_server_reset_action_val
*
* YANG 1.1 action validate callback
* Path: /server/reset
*
* INPUTS:
*     see agt/agt_action.h for details
*     k_ parameters are ancestor list key values.
*
* RETURNS:
*     error status
********************************************************************/
status_t u_ex_action_server_reset_action_val (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    xml_node_t *methnode,
    val_value_t *actionval,
    const xmlChar *k_server_name)
{
    status_t res = NO_ERR;
    val_value_t *errorval = NULL;
```

```
    if (LOGDEBUG) {
        log_debug("\nEnter u_ex_action_server_reset_action_val for action
<reset>");
    }

    val_value_t *v_reset_msg_val = NULL;
    const xmlChar *v_reset_msg;

    if (actionval) {
        v_reset_msg_val = val_find_child(
            actionval,
            y_ex_action_M_ex_action,
            y_ex_action_N_reset_msg);
        if (v_reset_msg_val) {
            v_reset_msg = VAL_STRING(v_reset_msg_val);
        }
    }

    /* the validate function would check here if it is OK to
     * reset the server right now; if not an error would be
     * returned by setting res to the correct error status.
     * If the input parameter is the problem then set errorval
     * otherwise leave errorval NULL
     */

    if (res != NO_ERR) {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_OPERATION,
            res,
            methnode,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval);
    }
    return res;

} /* u_ex_action_server_reset_action_val */
```

7.5.6 Action Invoke Callback Function

The action invoke callback function is used to perform the data-specific action requested by the client session. Only 1 action invoke function can register for each YANG action statement. There is usually one of these callback functions for every 'action' statement in the YANG module associated with the SIL code.

The action invoke callback function is optional to use, although if no invoke callback is provided, then the action will have no affect. Normally, this is only the case if the module is be tested by an application developer, using **netconfd-pro** as a server simulator.

It is enabled with the **agt_action_register_action** function, within the phase 1 initialization callback function.

The **yangdump-sdk** code generator will create this SIL callback function by default. There will be C comments in the code to indicate where your additional C code should be added.

The **val_find_child** function is commonly used to retrieve particular parameters within the RPC input section, which is encoded as a **val_value_t** tree. The **rpc_user1** and **rpc_user2** cache pointers in the **rpc_msg_t** structure can also be used to store data in the validation phase, so it can be immediately available in the invoke phase.

The **agt_record_error** function is commonly used to record any internal or platform-specific errors. In the **libtoaster** example, if the request to create a timer callback control block fails, then an error is recorded.

For RPC operations that return either an <ok> or <rpc-error> response, there is nothing more required of the RPC invoke callback function.

For operations which return some data or <rpc-error>, the SIL code must do 1 of 2 additional tasks:

- add a **val_value_t** structure to the **rpc_dataQ** queue in the **rpc_msg_t** for each parameter listed in the YANG rpc 'output' section.
- set the rpc_datacb pointer in the rpc_msg_t structure to the address of your data reply callback function. See the **agt_rpc_data_cb_t** definition in **agt/agt_rpc.h** for more details.

Example SIL Function Registration

```
   res = agt_action_register_action(
       (const xmlChar *)"/exa:server/exa:reset",
       AGT_RPC_PH_INVOKE,
       ex_action_server_reset_action_inv);
   if (res != NO_ERR) {
       return res;
   }
```

Example SIL Function:

(The **bold** text added after SIL stub generate).

```
/********************************************************************
* FUNCTION u_ex_action_server_reset_action_inv
*
* YANG 1.1 action invoke callback
* Path: /server/reset
*
* INPUTS:
```

```
*     see agt/agt_action.h for details
*     k_ parameters are ancestor list key values.
*
* RETURNS:
*     error status
********************************************************************/
status_t u_ex_action_server_reset_action_inv (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    xml_node_t *methnode,
    val_value_t *actionval,
    const xmlChar *k_server_name)
{
    status_t res = NO_ERR;

    if (LOGDEBUG) {
        log_debug("\nEnter u_ex_action_server_reset_action_inv for action
<reset>");
    }

    val_value_t *v_reset_msg_val = NULL;
    const xmlChar *v_reset_msg = NULL;

    if (actionval) {
        v_reset_msg_val = val_find_child(
            actionval,
            y_ex_action_M_ex_action,
            y_ex_action_N_reset_msg);
        if (v_reset_msg_val) {
            v_reset_msg = VAL_STRING(v_reset_msg_val);
        }
    }

    /* display the reset logging message */
    log_info("\nex-action: Resetting server %s\n", k_server_name);
    if (v_reset_msg) {
        log_info_append("%s\n", v_reset_msg);
    }

    /* the invoke function would schedule or execute the server reset here */

    /* remove the next line if scb is used */
    (void)scb;

    /* remove the next line if methnode is used */
    (void)methnode;

    /* invoke your device instrumentation code here */

    /* Following output nodes expected:
     * leaf reset-finished-at
     */
    if (actionval == NULL) {
        return ERR_NCX_OPERATION_FAILED;  // should not happen
    }

    obj_template_t *obj =
        obj_find_child(VAL_OBJ(actionval),
                       NULL,
                       NCX_EL_OUTPUT);
    if (obj == NULL) {
        return ERR_NCX_DEF_NOT_FOUND;  // should not happen
    }
```

```
    /* return the current time */
    xmlChar buff[TSTAMP_MIN_SIZE+1];
    tstamp_datetime(buff);

    val_value_t *val =
        agt_make_leaf(obj,
                      y_ex_action_N_reset_finished_at,
                      buff,
                      &res);
    if (retval) {
        agt_rpc_add_return_val(val, msg);
    } // else res set to error

    return res;

} /* u_ex_action_server_reset_action_inv */
```

7.5.7 Action Data Output Handling

YANG actions can return data to the client if the operation succeeds. The YANG “output” statement defines the return data for each YANG action. Constructing YANG data is covered in detail elsewhere. This section shows a simple example SIL action invoke function that returns data.

The output data is usually constructed with APIs like **agt_make_leaf** from **agt_util.h**.

To use these APIs, the output object is needed.

The following code can be used to get this object:

```
if (actionval == NULL) {
    return ERR_NCX_OPERATION_FAILED;  // should not happen
}

obj_template_t *obj =
    obj_find_child(VAL_OBJ(actionval),
                   NULL,
                   NCX_EL_OUTPUT);
if (obj == NULL) {
    return ERR_NCX_DEF_NOT_FOUND;  // should not happen
}
```

Once this object template is retrieved from the **actionval** parameter data can be added to the msg using **val_value_t** structures.

The following snippet shows a leaf being created using a **string** value.

```
xmlChar buff[TSTAMP_MIN_SIZE+1];
tstamp_datetime(buff);

val_value_t *val =
    agt_make_leaf(obj,
                  y_ex_action_N_reset_finished_at,
                  buff,
                  &res);
```

After the value is created it must be added to the message using the **agt_rpc_add_return_val** API:

```
if (val) {
    agt_rpc_add_return_val(val, msg);
}
```

7.6 Database Operations

The server database is designed so that the SIL callback functions do not need to really know which database model is being used by the server (E.g., target is **<candidate>** vs. **<running>** configuration).

There are three SIL database edit callback phases:

1. **Validate**: Check the parameters no matter what database is the target
2. **Apply**: The server will manipulate the database nodes as needed. The SIL usually has nothing to do in this phase unless internal resources need to be reserved.
3. **Commit or Rollback**: Depending on the result of the previous phases, either the commit or the rollback callback phase will be invoked, if and when the changes are going to be finalized in the running configuration.

The SIL code is not responsible for maintaining the value tree for any database. This is done by the server.

The SIL database edit callback code is responsible for the following tasks:

- Perform any data-model specific validation that is not already covered by a machine-readable statement, during the validation phase.
- Reserve any data-model specific resources for the proposed new configuration content, during the apply phase.
- Activate any data-model behavior changes based on the new configuration content, during the commit phase.
- Release any reserved resources that were previously allocated in the apply phase, during the rollback phase.

7.6.1 EDIT2 Callbacks

The server supports 2 modes of database editing callbacks. The original mode (EDIT1) is designed to invoke data node callbacks at the leaf level. This means that each altered leaf will cause a separate SIL callback. If no leaf callbacks are present, then the parent node will be invoked multiple times.

The EDIT2 mode is “list-based” or “container-based” instead.

- In this mode there are no SIL callback functions expected for terminal nodes (leaf, leaf-list, anyxml).
- The SIL callback for a container or list will be invoked, even if only child terminal nodes are being changed.
- The parent SIL callback will be invoked only once (per phase) if multiple child nodes are being altered at once
- The parent node for such an edit is flagged in the “undo” record as a special “edit2_merge”. The edit operation will be “OP_EDITOP_MERGE”, but the parent node is not being changed
- The special “edit2_merge” type of edit will have a queue of child_undo records containing info on the child edits. For example, 1 leaf could be created, another leaf modified, and a third leaf deleted, all in the same edit request. The **child_undo** records provide the edit operation and values being changed.

To use EDIT2 mode, simply register the SIL callback function with **agt_cb_register_edit2_callback** instead of **agt_cb_register_callback**.

7.6.2 SIL-SA EDIT2 Callbacks

The **SIL-SA EDIT2** callback usage is the same as the **EDIT2** callback except **EDIT2** MERGE handling. The difference is only in the children edits access APIs.

SIL-SA EDIT2 mode, the same way as **SIL** version, provides extended edit functionality and thus more ways to manage specific edit.

There are **SIL-SA EDIT2** mode specific high-level Transaction access and management utilities in **netconf/src/sil-sa/sil_sa.h**. These functions access the lower-level functions to provide simpler functions for common transaction management tasks.

The following table highlights available functions and SIL vs SIL-SA difference:

SIL-SA	SIL	Description
sil_sa_first_child_edit()	*agt_cfg_first_child_edit()*	Get the first child node edit record for a given transaction.
sil_sa_next_child_edit()	*agt_cfg_next_child_edit()*	Get the child edit fields from the undo record.
sil_sa_child_edit_fields()	*agt_cfg_child_edit_fields()*	Get the child edit fields from the specified undo record.

Note that the access APIs are different. This is the only difference between **SIL-SA** and **SIL** version of the **EDIT2** callbacks. Refer to the **EDIT2** callback section for more details on how to use **EDIT2** callbacks.

7.6.3 Database Template (cfg_template_t)

Every NETCONF database has a common template control block, and common set of access functions.

NETCONF databases are not separate entities like separate SQL databases. Each NETCONF database is conceptually the same database, but in different states:

- **candidate**: A complete configuration that may contain changes that have not been applied yet. This is only available if the **:candidate** capability is advertised by the server. The value tree for this database contains only configuration data nodes.
- **running**: The complete current server configuration. This is available on every NETCONF server. The value tree for this database contains configuration data nodes and non-configuration nodes created and maintained by the server. The server will maintain read-only nodes when **<edit-config>**, **<copy-config>,** or **<commit>** operations are performed on the running configuration. SIL code should not alter the data nodes within a configuration directly. This work is handled by the server. SIL callback code should only alter its own data structures, if needed.
- **startup**: A complete configuration that will be used upon the next reboot of the device. This is only available if the :startup capability is advertised by the server. The value tree for this database contains only configuration data nodes.

NETCONF also recognized external files via the <url> parameter, if the **:url** capability is advertised by the server. These databases will be supported in a future release of the server. The NETCONF standard does not require that these external databases support the same set of protocol operations as the standard databases, listed above. A client application can reliably copy from and to an external database, but editing and filtered retrieval may not be supported.

The following typedef is used to define a NETCONF database template:

```
/* struct representing 1 configuration database */
typedef struct cfg_template_t_ {
    dlq_hdr_t      qhdr;
    ncx_cfg_t      cfg_id;
    cfg_location_t cfg_loc;
    cfg_state_t    cfg_state;
    xmlChar       *name;
    xmlChar       *src_url;
    xmlChar       *load_time;
    xmlChar       *lock_time;
    xmlChar       *last_ch_time;
    uint32         flags;
    ses_id_t       locked_by;
    cfg_source_t   lock_src;
    dlq_hdr_t      load_errQ;    /* Q of rpc_err_rec_t */
    val_value_t   *root;          /* btyp == container */
} cfg_template_t;
```

The following table highlights the fields in the **cfg_template_t** data structure:

cfg_template_t Fields

Field	Description
qhdr	Queue header to allow the object template to be stored in a queue.

Field	Description
cfg_id	Internal configuration ID assigned to this configuration.
cfg_loc	Enumeration identifying the configuration source location.
cfg_state	Current internal configuration state.
name	Name string for this configuration.
src_url	URL for use with 'cfg_loc' to identify the configuration source.
load_time	Date and time string when the configuration was loaded.
lock_time	Date and time string when the configuration was last locked.
last_ch_time	Date and time string when the configuration was last changed.
flags	Internal configuration flags. Do not use directly.
locked_by	Session ID that owns the global configuration lock, if the database is currently locked.
lock_src	If the database is locked, identifies the protocol or other source that currently caused the database to be locked.
load_errQ	Queue of rpc_err_rec_t structures that represent any <rpc-error> records that were generated when the configuration was loaded, if any.
root	The root of the value tree representing the entire database.

7.6.4 Database Access Functions

The file **ncx/cfg.h** contains some high-level database access functions that may be of interest to SIL callback functions for custom RPC operations. All database access details are handled by the server if the database edit callback functions are used (associated with a particular object node supported by the server). The following table highlights the most commonly used functions. Refer to the H file for a complete definition of each API function.

cfg_template_t Access Functions

Function	Description
cfg_new_template	Create a new configuration database.
cfg_free_template	Free a configuration database.
cfg_get_state	Get the current internal database state.
cfg_get_config	Get a configuration database template pointer, from a configuration name string.
cfg_get_config_name	Get the config name from its ID.
cfg_get_config_id	Get a configuration database template pointer, from a configuration ID.
cfg_set_target	Set the CFG_FL_TARGET flag in the specified config.
cfg_fill_candidate_from_running	Fill the <candidate> config with the config contents of the <running> config.
cfg_fill_candidate_from_startup	Fill the <candidate> config with the config contents of the <startup> config.
cfg_fill_candidate_from_inline	Fill the candidate database from an internal value tree data structure.
cfg_get_dirty_flag	Returns TRUE if the database has changes in it that have not been saved yet. This applies to the candidate and running databases at this time.
cfg_clear_dirty_flag	Clear the cfg dirty flag upon request.
cfg_clear_running_dirty_flag	Clear the running dirty flag when it is saved to NV-storage or loaded into running from startup.
cfg_clear_candidate_dirty_flag	Clear the candidate dirty flag when it is saved to NV-storage or loaded into running from startup.
cfg_get_dirty_flag	Get the config dirty flag value.
cfg_ok_to_lock	Check if the database could be successfully locked by a specific session.
cfg_ok_to_unlock	Check if the database could be successfully unlocked by a specific session.
cfg_ok_to_read	Check if the database is in a state where read operations are allowed.
cfg_ok_to_write	Check if the database could be successfully written by a specific session. Checks the global configuration lock, if any is set.
cfg_is_global_locked	Returns TRUE if the database is locked right now with a global lock.

Function	Description
cfg_get_global_lock_info	Get some information about the current global lock on the database.
cfg_is_partial_locked	Check if the specified config has any active partial locks.
cfg_add_partial_lock	Add a partial lock the specified config. This will not really have an effect unless the CFG_FL_TARGET flag in the specified config is also set . For global lock only.
cfg_find_partial_lock	Find a partial lock in the specified config.
cfg_first_partial_lock	Get the first partial lock in the specified config.
cfg_next_partial_lock	Get the next partial lock in the specified config.
cfg_delete_partial_lock	Remove a partial lock from the specified config.
cfg_ok_to_partial_lock	Check if the specified config can be locked right now for partial lock only.
cfg_get_root	Get the config root for the specified config.
cfg_lock	Get a global lock on the database.
cfg_unlock	Release the global lock on the database.
cfg_unlock_ex	Release the global lock on the database. Do not always force a remove changes. This is needed for the <unload> operation which locks the datastores while deleting data nodes and schema nodes for the module being unloaded.
cfg_release_locks	Release all locks on all databases.
cfg_release_partial_locks	Release any configuration locks held by the specified session.
cfg_get_lock_list	Get a list of all the locks held by a session.
cfg_update_last_ch_time	Update the last-modified time-stamp.
cfg_update_last_txid	Update the last good transaction ID.
cfg_update_stamps	Update the last-modified and last-txid stamps.
cfg_get_last_ch_time	Get the last-modified time-stamp.
cfg_get_last_txid	Get the last good transaction ID.
cfg_sprintf_etag	Write the Entity Tag for the datastore to the specified buffer.
cfg_get_startup_filespec	Get the filespec string for the XML file to save the running database.

7.6.5 Database Callback Initialization and Cleanup

The file **agt/agt_cb.h** contains functions that a SIL or SIL-SA developer needs to register and unregister database edit callback functions. The same callback function can be used for different phases, if desired.

The following function template definition is used for all SIL or SIL-SA database edit callback functions:

```
/* Callback function for agent object handler
 * Used to provide a callback sub-mode for
 * a specific named object
 *
 * INPUTS:
 *   scb == session control block making the request
 *   msg == incoming rpc_msg_t in progress
 *   cbtyp == reason for the callback
 *   editop == the parent edit-config operation type, which
 *             is also used for all other callbacks
 *             that operate on objects
 *   newval == container object holding the proposed changes to
 *          apply to the current config, depending on
 *          the editop value. Will not be NULL.
 *   curval == current container values from the <running>
 *          or <candidate> configuration, if any. Could be NULL
 *          for create and other operations.
 *
 * RETURNS:
 *    status:
 */
typedef status_t
    (*agt_cb_fn_t) (ses_cb_t  *scb,
                    rpc_msg_t *msg,
                    agt_cbtyp_t cbtyp,
                    op_editop_t  editop,
                    val_value_t  *newval,
                    val_value_t  *curval);
```

SIL or SIL-SA Database Callback Template

Parameter	Description
scb	The session control block making the edit request. The SIL callback code should not need this parameter except to pass to functions that need the SCB.
msg	Incoming RPC message in progress. The server uses some fields in this structure, and there are 2 SIL fields for the RPC callback functions. The SIL callback code should not need this parameter except to pass to functions that need the message header.
cbtyp	Enumeration for the callback phase in progress..
editop	The edit operation in effect as this node is being processed.
newval	Value node containing the new value for a create, merge, replace, or insert operation. The 'newval' parm

Parameter	Description
	may be NULL and should be ignored for a delete operation. In that case, the 'curval' pointer contains the node being deleted.
curval	Value node containing the current database node that corresponds to the 'newval' node, if any is available.

A SIL or SIL-SA database edit callback function is hooked into the server with the **agt_cb_register_callback** or **agt_cb_register_edit2_callback** functions, described below. The SIL or SIL-SA code generated by yangdump-pro uses the first function to register a single callback function for all callback phases.

The **agt_cb_register_edit2_callback** function is used to declare the callback as an “edit2” style callback.

```
extern status_t
    agt_cb_register_callback (const xmlChar *modname,
                              const xmlChar *defpath,
                              const xmlChar *version,
                              const agt_cb_fn_t cbfn);
```

agt_cb_register_callback

Parameter	Description
modname	Module name string that defines this object node.
defpath	Absolute path expression string indicating which node the callback function is for.
version	If non-NULL, indicates the exact module version expected.
cbfn	The callback function address. This function will be used for all callback phases.

```
extern status_t
    agt_cb_register_edit2_callback (const xmlChar *modname,
                                    const xmlChar *defpath,
                                    const xmlChar *version,
                                    const agt_cb_fn_t cbfn);
```

agt_cb_register_edit2_callback

Parameter	Description
modname	Module name string that defines this object node.
defpath	Absolute path expression string indicating which node the callback function is for.
version	If non-NULL, indicates the exact module version expected.

Parameter	Description
cbfn	The callback function address. This function will be used for all callback phases.

The **agt_cb_unregister_callbacks** function is called during the module cleanup. It is generated by **yangdump-pro** automatically for all RPC operations.

```
extern void
    agt_cb_unregister_callbacks (const xmlChar *modname,
                                 const xmlChar *defpath);
```

agt_cb_unregister_callbacks

Parameter	Description
modname	Module name string that defines this object node.
defpath	Absolute path expression string indicating which node the callback function is for.

7.6.6 Example SIL Database Edit Callback Function

The following example code is from the **libtoaster** source code, available in **yumapro-dev** and **yumapro-source** packages.

```
/********************************************************************
* FUNCTION y_toaster_toaster_edit
*
* Edit database object callback
* Path: /toaster
* Add object instrumentation in COMMIT phase.
*
* INPUTS:
*     see agt/agt_cb.h for details
*
* RETURNS:
*     error status
********************************************************************/

static status_t
    y_toaster_toaster_edit (
        ses_cb_t *scb,
        rpc_msg_t *msg,
        agt_cbtyp_t cbtyp,
        op_editop_t editop,
        val_value_t *newval,
        val_value_t *curval)
{
    status_t res;
    val_value_t *errorval;
    const xmlChar *errorstr;

    res = NO_ERR;
    errorval = NULL;
    errorstr = NULL;

    switch (cbtyp) {
    case AGT_CB_VALIDATE:
        /* description-stmt validation here */
        break;
    case AGT_CB_APPLY:
        /* database manipulation done here */
        break;
    case AGT_CB_COMMIT:
        /* device instrumentation done here */
        switch (editop) {
        case OP_EDITOP_LOAD:
            toaster_enabled = TRUE;
            toaster_toasting = FALSE;
            break;
        case OP_EDITOP_MERGE:
            break;
        case OP_EDITOP_REPLACE:
            break;
        case OP_EDITOP_CREATE:
            toaster_enabled = TRUE;
            toaster_toasting = FALSE;
            break;
        case OP_EDITOP_DELETE:
            toaster_enabled = FALSE;
```

```
            if (toaster_toasting) {
                agt_timer_delete(toaster_timer_id);
                toaster_timer_id = 0;
                toaster_toasting = FALSE;
                y_toaster_toastDone_send((const xmlChar *)"error");
            }
            break;
        default:
            res = SET_ERROR(ERR_INTERNAL_VAL);
        }

        if (res == NO_ERR) {
            res = agt_check_cache(
                &toaster_val,
                newval,
                curval,
                editop);
        }

       if (res == NO_ERR &&
            (editop == OP_EDITOP_LOAD || editop == OP_EDITOP_CREATE)){
            res = y_toaster_toaster_mro(newval);
        }
        break;
    case AGT_CB_ROLLBACK:
        /* undo device instrumentation here */
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    /* if error: set the res, errorstr, and errorval parms */
    if (res != NO_ERR) {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_CONTENT,
            res,
            NULL,
            NCX_NT_STRING,
            errorstr,
            NCX_NT_VAL,
            errorval);
    }

    return res;

} /* y_toaster_toaster_edit */
```

7.6.7 EDIT2 Callback Function Example

The following sections illustrates how to utilize the **EDIT2** callbacks in examples.

The **EDIT2** callback template is the same as the **EDIT1** callback. The difference is that there is a queue of child edit records that may need to be accessed to reliably process the edit requests.

In the following example, **EDIT2** callback code gets each **child_undo** record in order to retrieve the real edited nodes and the edited operation and merely form the data and prints it to the log. Instead of printing the data an agent could process to the next step and run device instrumentation as required.

The following example code illustrates how the **EDIT2** callback may look like.

```
/********************************************************************
* FUNCTION  edit2_callback_example
*
* Callback function for server object handler
* Used to provide a callback sub-mode for
* a specific named object
*
* Path: /interfaces/interface
* Add object instrumentation in COMMIT phase.
*
*********************************************************************/
    edit2_callback_example (ses_cb_t *scb,
                            rpc_msg_t *msg,
                            agt_cbtyp_t cbtyp,
                            op_editop_t editop,
                            val_value_t *newval,
                            val_value_t *curval)
{
    status_t res = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    if (LOGDEBUG) {
        log_debug("\nEnter edit2_callback_example callback for %s phase",
            agt_cbtype_name(cbtyp));
    }

    switch (cbtyp) {
    case AGT_CB_VALIDATE:
        /* description-stmt validation here */
        break;
    case AGT_CB_APPLY:
        /* database manipulation done here */
        break;
    case AGT_CB_COMMIT:
        /* device instrumentation done here */
        switch (editop) {
        case OP_EDITOP_LOAD:
            break;
        case OP_EDITOP_MERGE:
            /* the edit is not really on this node; need to get
             * each child_undo record to get the real edited nodes
             * and the edited operations
             */
            agt_cfg_transaction_t *txcb = RPC_MSG_TXCB(msg);
            agt_cfg_undo_rec_t *child_edit =
                agt_cfg_first_child_edit(txcb, newval, curval);
```

```
        while (child_edit) {
            op_editop_t child_editop = OP_EDITOP_NONE;
            val_value_t *child_newval = NULL;
            val_value_t *child_curval = NULL;
            xmlChar *newval_str = NULL;
            xmlChar *curval_str = NULL;

            agt_cfg_child_edit_fields(child_edit,
                                      &child_editop,
                                      &child_newval,
                                      &child_curval);

            if (child_newval) {
                newval_str = val_make_sprintf_string(child_newval);
                if (newval_str == NULL) {
                    return ERR_INTERNAL_MEM;
                }
            }
            if (child_curval) {
                curval_str = val_make_sprintf_string(child_curval);
                if (curval_str == NULL) {
                    m__free(newval_str);
                    return ERR_INTERNAL_MEM;
                }
            }

            log_info("\n        %s: editop=%s newval=%s curval=%s",
                  child_newval ? VAL_NAME(child_newval) : NCX_EL_NULL,
                  child_editop ? op_editop_name(child_editop) : NCX_EL_NONE,
                  child_newval ? newval_str : NCX_EL_NULL,
                  child_curval ? curval_str : NCX_EL_NULL);


            /* Force Rollback if the child value is not acceptable */
            if (child_newval &&
                !xml_strcmp(VAL_NAME(child_newval),
                            (const xmlChar *)"untagged-ports") &&
                !xml_strcmp(VAL_STR(child_newval),
                            (const xmlChar *)"not-supported")) {

                res = ERR_NCX_OPERATION_NOT_SUPPORTED;
                m__free(newval_str);
                m__free(curval_str);
                break;
            }

            /**** process child edits here ****/


            m__free(newval_str);
            m__free(curval_str);

            child_edit = agt_cfg_next_child_edit(child_edit);
        }

        break;
    case OP_EDITOP_REPLACE:
    case OP_EDITOP_CREATE:
        /* the edit is on this list node and the child editop
         * can be treated the same as the parent (even if different)
         * the val_value_t child nodes can be accessed and there
         * are no child_undo records to use
```

```
             */
            val_value_t *child_newval = NULL;
            child_newval =
                val_find_child(newval,
                               EXAMPLE_MODNAME,
                               (const xmlChar *)"untagged-ports");

                val_value_t *leaflist_val = child_newval;
                while (leaflist_val) {

                    /**** process child leaf-list edits here ****/

                    leaflist_val = val_next_child_same(leaflist_val);
                }

                /**** process other child edits here if needed ****/

            break;
        case OP_EDITOP_DELETE:
            break;
        default:
            res = SET_ERROR(ERR_INTERNAL_VAL);
        }
        break;
    case AGT_CB_ROLLBACK:
        /* undo device instrumentation here */
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(scb,
                         &msg->mhdr,
                         NCX_LAYER_CONTENT,
                         res,
                         NULL,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval);
    }

    return res;

} /* edit2_callback_example */
```

NOTE:

Do not use **SET_ERROR**() macro to return normal errors. This macro will cause "**assert**()" to be invoked, halting program execution. The macro defined in **netconf/src/ncx/status.h** and it is for flagging internal errors only. This macro must not be used for normal errors.

7.6.8 SIL-SA EDIT2 Callback Function Example

The following sections illustrates how to utilize the **SIL-SA** version of the **EDIT2** callbacks in examples.

The **SIL-SA EDIT2** callback usage is the same as the **EDIT2** callback except **EDIT2** MERGE handling. The difference is only in the children edits access APIs.

In the following example, **SIL-SA EDIT2** callback code gets each **child edit** record in order to retrieve the real edited nodes and the edited operation and merely form the data and prints it to the log. Instead of printing the data an agent could process to the next step and run device instrumentation as required.

Note that the access APIs are different. This is the only difference between **SIL-SA** and **SIL** version of the **EDIT2** callbacks. Refer to the **EDIT2** callback section for more details on how to use **EDIT2** callbacks.

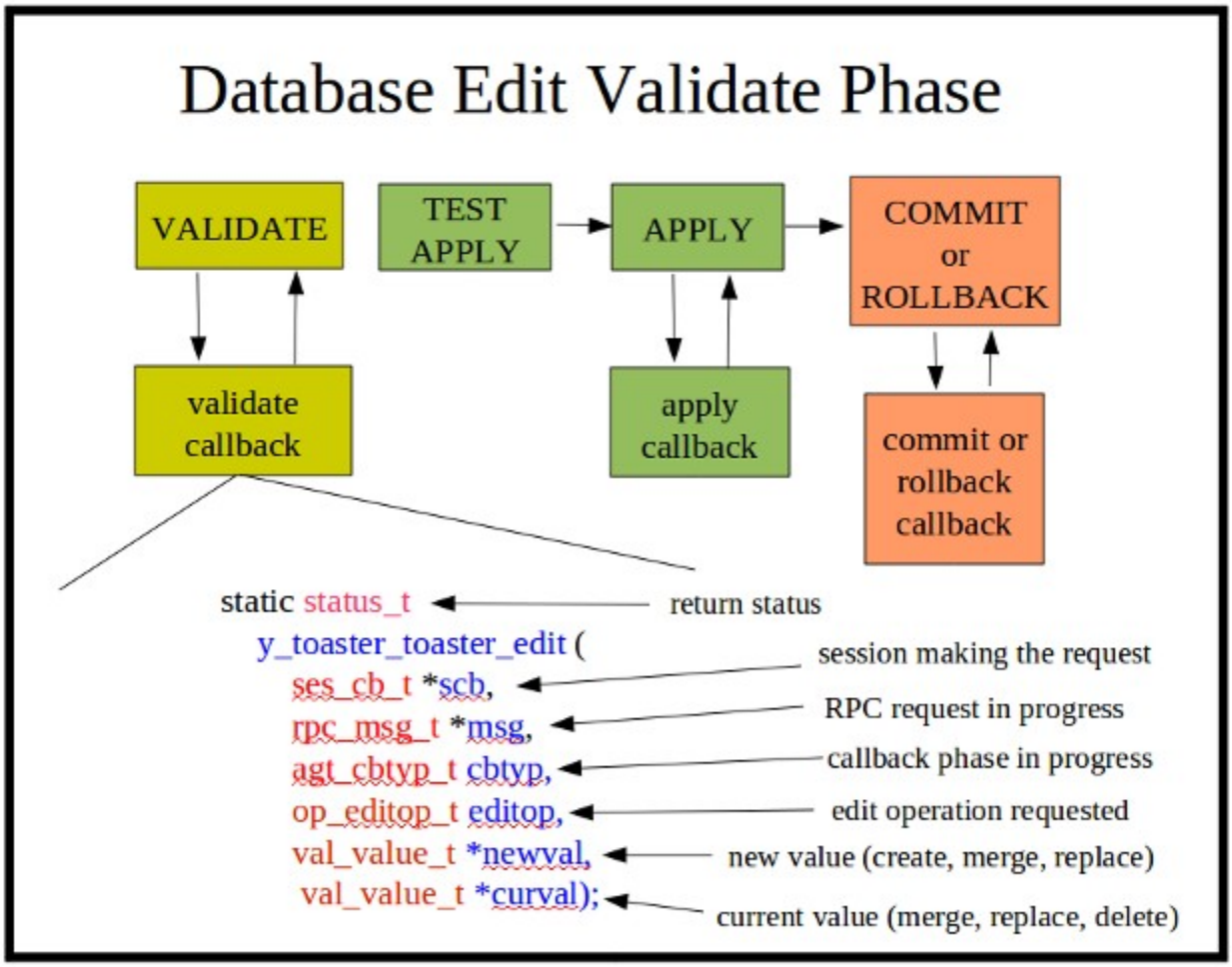
The following example code illustrates how the **SIL-SA EDIT2** callback may look like.

```
/********************************************************************
* FUNCTION  silsa_edit2_callback_example
*
* SIL-SA EDIT2 Callback function for server object handler
* Used to provide a callback sub-mode for
* a specific named object
*
* Path: /interfaces/interface
* Add object instrumentation in COMMIT phase.
*
*********************************************************************/
    silsa_edit2_callback_example (ses_cb_t *scb,
                                  rpc_msg_t *msg,
                                  agt_cbtyp_t cbtyp,
                                  op_editop_t editop,
                                  val_value_t *newval,
                                  val_value_t *curval)
{
    status_t res = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    if (LOGDEBUG) {
        log_debug("\nEnter silsa_edit2_callback_example callback for %s phase",
            agt_cbtype_name(cbtyp));
    }

    switch (cbtyp) {
    case AGT_CB_VALIDATE:
        /* description-stmt validation here */
        break;
    case AGT_CB_APPLY:
        /* database manipulation done here */
        break;
    case AGT_CB_COMMIT:
        /* device instrumentation done here */
        switch (editop) {
        case OP_EDITOP_LOAD:
            break;
        case OP_EDITOP_MERGE:
            /* the edit is not really on this node; need to get
             * each child_undo record to get the real edited nodes
             * and the edited operations
             */
```

```
        sil_sa_child_edit_t *child_edit = sil_sa_first_child_edit(msg);
        while (child_edit) {

            op_editop_t child_editop = OP_EDITOP_NONE;
            val_value_t *child_newval = NULL;
            val_value_t *child_curval = NULL;
            xmlChar *newval_str = NULL;
            xmlChar *curval_str = NULL;

            sil_sa_child_edit_fields(child_edit,
                                     &child_editop,
                                     &child_newval,
                                     &child_curval);
            if (child_newval) {
                newval_str = val_make_sprintf_string(child_newval);
                if (newval_str == NULL) {
                    return;
                }
            }
            if (child_curval) {
                curval_str = val_make_sprintf_string(child_curval);
                if (curval_str == NULL) {
                    if (newval_str) {
                        m__free(newval_str);
                    }
                    return;
                }
            }

            log_debug("\n        %s: editop=%s newval=%s curval=%s",
                      child_newval ? VAL_NAME(child_newval) : NCX_EL_NULL,
                      child_editop ? op_editop_name(child_editop) : NCX_EL_NONE,
                      child_newval ? newval_str : NCX_EL_NULL,
                      child_curval ? curval_str : NCX_EL_NULL);

            m__free(newval_str);
            m__free(curval_str);

            child_edit = sil_sa_next_child_edit(child_edit);
        }

        break;
    case OP_EDITOP_REPLACE:
    case OP_EDITOP_CREATE:
        /* the edit is on this list node and the child editop
         * can be treated the same as the parent (even if different)
         * the val_value_t child nodes can be accessed and there
         * are no child_undo records to use
         */
        val_value_t *child_newval = NULL;
        child_newval =
            val_find_child(newval,
                           EXAMPLE_MODNAME,
                           (const xmlChar *)"untagged-ports");

            val_value_t *leaflist_val = child_newval;
            while (leaflist_val) {

                /**** process child leaf-list edits here ****/

                leaflist_val = val_next_child_same(leaflist_val);
            }
```

```
                /**** process other child edits here if needed ****/

            break;
        case OP_EDITOP_DELETE:
            break;
        default:
            res = SET_ERROR(ERR_INTERNAL_VAL);
        }
        break;
    case AGT_CB_ROLLBACK:
        /* undo device instrumentation here */
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(scb,
                         &msg->mhdr,
                         NCX_LAYER_CONTENT,
                         res,
                         NULL,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval);
    }

    return res;

} /* silsa_edit2_callback_example */
```

7.6.9 Database Edit Validate Callback Phase



A SIL database validation phase callback function is responsible for checking all the 'description statement' sort of data model requirements that are not covered by any of the YANG machine-readable statements.

For example, if a 'user name' parameter needed to match an existing user name in **/etc/passwd,** then the SIL validation callback would call the system APIs needed to check if the 'newval' string value matched a valid user name. The server will make sure the user name is well-formed and could be a valid user name.

7.6.10 Database Edit Apply Callback Phase

The callback function for this phase is called when database edits are being applied to the running configuration. The resources needed for the requested operation may be reserved at this time, if needed.

7.6.11 Database Edit Commit Callback Phase

This callback function for this phase is called when database edits are being committed to the running configuration. The SIL callback function is expected to finalize and apply any data-model dependent system behavior at this time.

7.6.12 Database Edit Rollback Callback Phase

This callback function for this phase is called when database edits are being undone, after some apply phase or commit phase callback function returned an error, or a confirmed commit operation timed out.

The SIL callback function is expected to release any resources it allocated during the apply or commit phases. Usually only the commit or the rollback function will be called for a given SIL callback, but it is possible for both to be called. For example, if the 'rollback-on-error' option is in effect, and some SIL commit callback fails after your SIL commit callback succeeds, then your SIL rollback callback may be called as well.

7.6.13 Database Virtual Node Get Callback Function

A common SIL callback function to use is a virtual node 'get' function. A virtual node can be either a configuration or non-configuration node, but is more likely to be a non-configuration node, such as a counter or hardware status object.

The function **agt_make_virtual_leaf** in **agt/agt_util.h** is a common API used for creating a virtual leaf within an existing parent container.

The following typedef defines the **getcb_fn_t** template, used by all virtual callback functions. This function is responsible for filling in a value node with the current instance value. The status NO_ERR is returned if this is done successfully.

```
/* getcb_fn_t
 *
 * Callback function for agent node get handler
 *
 * INPUTS:
 *   scb    == session that issued the get (may be NULL)
 *             can be used for access control purposes
 *   cbmode == reason for the callback
 *   virval == place-holder node in the data model for
 *              this virtual value node
 *   dstval == pointer to value output struct
 *
 * OUTPUTS:
 *  *fil may be adjusted depending on callback reason
 *  *dstval should be filled in, depending on the callback reason
 *
 * RETURNS:
 *    status:
 */
typedef status_t
    (*getcb_fn_t) (ses_cb_t *scb,
                   getcb_mode_t cbmode,
                   const val_value_t *virval,
                   val_value_t *dstval);
```

The following table describes the parameters.

getcb_fn_t Parameters

Parameter	Description
scb	This is the session control block making the request, if available. This pointer may be NULL, so in the rare event this parameter is needed by the SIL callback function, it should be checked first.
cbmode	The callback type. The only supported enumeration at this time is GETCB_GET_VALUE. Other values may be used in the future for different retrieval modes.
virval	The virtual value node in the data tree that is being referenced, The **val_get_virtual_value** function was called for this value node.
dstval	The destination value node that needs to be filled in. This is just an empty value node that has been malloced and then initialized with the **val_init_from_template** function. The SIL callback function needs to set the value properly. The **val_set_simval** and **val_set_simval_obj** functions are two API functions that can be used for this purpose.

The following example from **agt/agt_ses.c** shows a SIL get callback function for the ietf-netconf-monitoring data model, which returns the 'in-sessions' counter value:

```
/********************************************************************
* FUNCTION agt_ses_get_inSessions
*
* <get> operation handler for the inSessions counter
*
* INPUTS:
*    see ncx/getcb.h getcb_fn_t for details
*
* RETURNS:
*    status
********************************************************************/
status_t
    agt_ses_get_inSessions (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) {
        VAL_UINT(dstval) = agttotals->inSessions;
        return NO_ERR;
    } else {
        return ERR_NCX_OPERATION_NOT_SUPPORTED;
    }

} /* agt_ses_get_inSessions */
```

The following example from **agt/agt_state.c** shows a complex get callback function for the same data model, which returns the entire <capabilities> element when it is requested. This is done by simply cloning the 'official copy' of the server capabilities that is maintained by the **agt/agt_caps.c** module.

```
/********************************************************************
* FUNCTION get_caps
*
* <get> operation handler for the capabilities NP container
*
* INPUTS:
*    see ncx/getcb.h getcb_fn_t for details
*
* RETURNS:
*    status
********************************************************************/
static status_t
    get_caps (ses_cb_t *scb,
              getcb_mode_t cbmode,
              val_value_t *virval,
              val_value_t  *dstval)
{
    val_value_t       *capsval;
    status_t           res;

    (void)scb;
    (void)virval;
    res = NO_ERR;

    if (cbmode == GETCB_GET_VALUE) {
        capsval = val_clone(agt_cap_get_capsval());
        if (!capsval) {
            return ERR_INTERNAL_MEM;
        } else {
            /* change the namespace to this module,
             * and get rid of the netconf NSID
             */
            val_change_nsid(capsval, statemod->nsid);
            val_move_children(capsval, dstval);
            val_free_value(capsval);
        }
    } else {
        res = ERR_NCX_OPERATION_NOT_SUPPORTED;
    }
    return res;

}  /* get_caps */
```

7.6.14 Getting the Current Transaction ID

In order to determine if an edit transaction is in progress, an API can be used. If the transaction-id returned by this function is zero, then no transaction is in progress.

```
/*********************************************************************
* FUNCTION agt_cfg_txid_in_progress
*
* Return the ID of the current transaction ID in progress
*
* INPUTS:
*    cfgid == config ID to check
* RETURNS:
*    txid of transaction in progress or 0 if none
*********************************************************************/
extern ncx_transaction_id_t
    agt_cfg_txid_in_progress (ncx_cfg_t cfgid);
```

Example: Get the current transaction ID for the running datastore

```
   ncx_transaction_id_t txid = agt_cfg_txid_in_progress(NCX_CFGID_RUNNING);
```

7.6.15 Finding Data Nodes with XPath

The internal XPath API can be used to access configuration data nodes. The data should not be altered. This API should be considered read-only.

- Step 1) Create the XPath parser control block

```
      xpath_pcb_t *pcb =
          xpath_new_pcb((const xmlChar *)"/interfaces/interface, NULL);
       if (pcb == NULL) {
           return ERR_INTERNAL_MEM;
       }
```

- Step 2) Evaluate the XPath expression
  - **docroot** is usually a datastore root like **cfg->cfg_root**
  - context node can be an interior node, but can also be the **docroot**

```
       xpath_result_t *result =
           xpath1_eval_expr(pcb,
                            root,
                            root,
                            FALSE, // logerrors
                            TRUE,  // configonly
                            &res);
```

- Step 3) Iterate through the node-set result

```
   xpath_resnode_t *resnode = xpath_get_first_resnode(result);
    for (; resnode; resnode = xpath_get_next_resnode(resnode)) {
       val_value_t *valnode = xpath_get_resnode_valptr(resnode);

       // do something with valnode....
     }
```

```
/********************************************************************
* FUNCTION xpath_new_pcb
*
* malloc a new XPath parser control block
* xpathstr is allowed to be NULL, otherwise
* a strdup will be made and exprstr will be set
*
* Create and initialize an XPath parser control block
*
```

```
* INPUTS:
*   xpathstr == XPath expression string to save (a copy will be made)
*            == NULL if this step should be skipped
*   getvar_fn == callback function to retirieve an XPath
*                variable binding
*                NULL if no variables are used
*
* RETURNS:
*   pointer to malloced struct, NULL if malloc error
*********************************************************************/
extern xpath_pcb_t *
    xpath_new_pcb (const xmlChar *xpathstr,
                   xpath_getvar_fn_t  getvar_fn);


/********************************************************************
* FUNCTION xpath1_eval_expr
*
* use if the prefixes are YANG: must/when
* Evaluate the expression and get the expression nodeset result
*
* INPUTS:
*    pcb == XPath parser control block to use
*    val == start context node for value of current()
*    docroot == ptr to cfg->root or top of rpc/rpc-replay/notif tree
*    logerrors == TRUE if log_error and ncx_print_errormsg
*                 should be used to log XPath errors and warnings
*                 FALSE if internal error info should be recorded
*                 in the xpath_result_t struct instead
*    configonly ==
*           XP_SRC_XML:
*                 TRUE if this is a <get-config> call
*                 and all config=false nodes should be skipped
*                 FALSE if <get> call and non-config nodes
*                 will not be skipped
*           XP_SRC_YANG and XP_SRC_LEAFREF:
*                 should be set to false
*     res == address of return status
*
* OUTPUTS:
*   *res is set to the return status
*
*
* RETURNS:
*   malloced result struct with expr result
*   NULL if no result produced (see *res for reason)
*********************************************************************/
extern xpath_result_t *
    xpath1_eval_expr (xpath_pcb_t *pcb,
                      val_value_t *val,
                      val_value_t *docroot,
                      boolean logerrors,
                      boolean configonly,
                      status_t *res);


/********************************************************************
* FUNCTION xpath_get_first_resnode
*
* Get the first result in the renodeQ from a result struct
*
* INPUTS:
*    result == result struct to check
```

```
*
* RETURNS:
*   pointer to resnode or NULL if some error
*********************************************************************/
extern xpath_resnode_t *
    xpath_get_first_resnode (xpath_result_t *result);


/*********************************************************************
* FUNCTION xpath_get_next_resnode
*
* Get the first result in the renodeQ from a result struct
*
* INPUTS:
*    result == result struct to check
*
* RETURNS:
*   pointer to resnode or NULL if some error
*********************************************************************/
extern xpath_resnode_t *
    xpath_get_next_resnode (xpath_resnode_t *resnode);


/*********************************************************************
* FUNCTION xpath_get_resnode_valptr
*
* Get the first result in the renodeQ from a result struct
*
* INPUTS:
*    result == result struct to check
*
* RETURNS:
*   pointer to resnode or NULL if some error
*********************************************************************/
extern val_value_t *
    xpath_get_resnode_valptr (xpath_resnode_t *resnode);
```

7.6.16 Determining if a Value Node is Set

A **val_value_t** structure is usually allocated with **val_new_value()** and then initialized with **val_init_from_template().**

In order to determine if the value has been set by a client, and not set by default or simply initialized, use the API function **val_is_value_set().**

```
/********************************************************************
* FUNCTION val_is_value_set
*
* Check if a value has been set by a client
* It has to be initialized and not set by default to return true
*
* INPUTS:
*   val == value to check
* RETURNS:
*   true if value has been set
*   false if has not been set, or set to default, or the code
*   cannot tell if the value is more than initialized
*********************************************************************/
extern boolean
    val_is_value_set (val_value_t *val);
```

7.6.17 Determining if the SIL Callback is the Deepest for the Edit

A datastore edit includes data for a subtree of configuration data. It is possible that multiple SIL callbacks will be invoked for the edit operation.

The function **agt_val_silcall_is_deepest()** from file **agt/agt_val_silcall.c** can be used to determine if the current callback is the deepest SIL invocation for the edit.

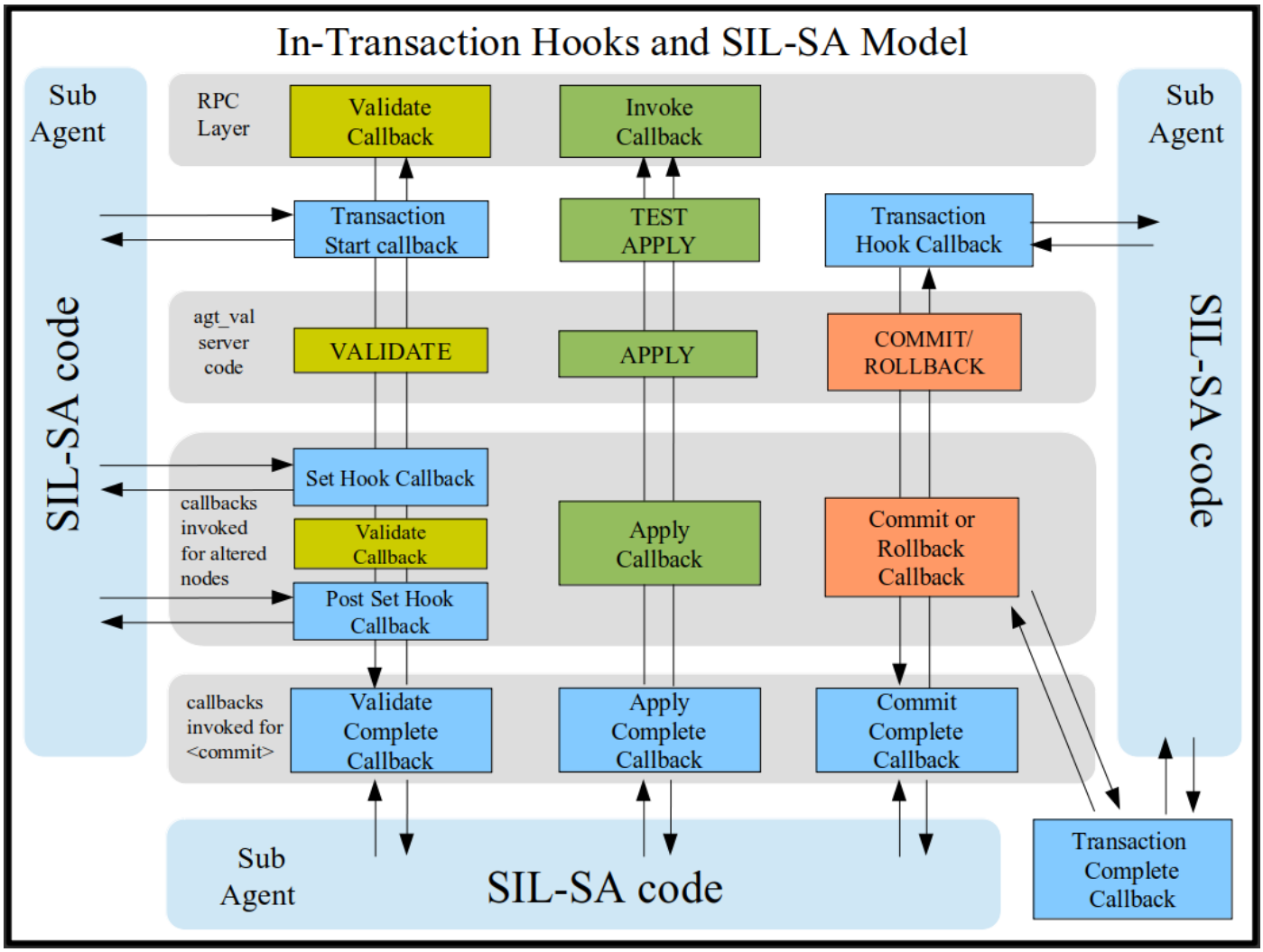
```
/********************************************************************
* FUNCTION agt_val_silcall_is_deepest
*
* Check if the current nested SIL callback is at the deepest
* level for the current edit.
*
  container tnest {
    list tlist {
      key idx;
      leaf idx { type int32; }
      list tlist2 {
        key idx2;
        leaf idx2 { type int32; }
        leaf c1 { type string; }
        list tlist3 {
          key idx3;
          leaf idx3 { type int32; }
          leaf c2 { type string; }
        }
      }
```

```
    }
  }

* In the example above, the code is generated as EDIT2 code
* so there are no callbacks for leafs
*
* This function would return TRUE for list 'tlist3' and
* FALSE for all ancestor node callbacks (tnest, tlist, list2)
*
* INPUTS:
*    msg == RPC message to check
* RETURNS:
*    TRUE if the current SIL callback is at the deepest level
*    for the edit that contains the silcall record
*
*    FALSE if the current SIL callback is not at the deepest level
*    for the edit that contains the silcall record. There is at least
*    one nested_silcall involved in the current edit that is at
*    a child level compared to the current node being called
*
*    FALSE if there is no current silcall set
*********************************************************************/
extern boolean
    agt_val_silcall_is_deepest (rpc_msg_t  *msg);
```

7.7 Database Editing with In-Transaction APIs

In-transaction capabilities are being added to the **netconfd-pro** server API to allow an application to update the configuration data and perform additional actions while the transaction is in progress. This can be a vital solution to implement many services in switch/router management operations.

The following diagram illustrates what callbacks are invoked and when to expect them to be invoked during the transaction.



The following callback APIs are supported:

- **Set Hook:** Similar to the **EDIT1** or **EDIT2** callbacks, except this callback is only invoked once at the start of the edit for a particular data node. This callback can alter the data in the current edit and also add new edits to the current transaction. This callback will be invoked before **EDIT1** or **EDIT2** callbacks for the same object.
- **Post Set Hook:** Similar to the **Set Hook** callback except that it is invoked when an object is modified but **AFTER EDIT** callback is done for the same object. When the **netconfd-pro** server has been configured to provide a candidate configuration, **Post Set Hook** code will be invoked when changes are done to the **<candidate>** configuration if the **--target=candidate** parameter is used. If **--target=running** then the **Post Set Hoo**k will be invoked at the start of the transaction on the **<running>** datastore. This callback will be invoked **AFTER** a **EDIT1** or **EDIT2** callback for the same object.

- **Transaction Hook**: Similar to the **Set Hook** except this callback is invoked just before the data is committed to the running datastore. This callback will be invoked after **EDIT1** or **EDIT2** callbacks for the same object.
- **Transaction Start:** The callback that is intended to provide access to the transaction control block and its parameters right before the transaction is started.
- **Transaction Complete:** The callback that is intended to provide access to the transaction control block and its parameters right after the transaction is completed. Called for every transaction completion, not just transactions that complete without errors.
- **Validate Complete:** The callback is intended to allow manipulations with the running and candidate configurations (equivalent to completion of validation phase during <commit>).
- **Apply Complete:** The callback is intended to allow manipulations with the running and candidate configurations (equivalent to completion of apply phase during <commit>).
- **Commit Complete:** The callback is a user/system callback that is invoked when the <commit> operation completes without errors or the internal <replay-config> operation completes without errors.
- **Rollback Complete:** The callback is the user/system callback that is invoked after and if the Rollback Phase has been processed during the <commit> operation.
- **Set Order Hook**: Set the secondary SIL priority for instances of the same list object that is being set in the same edit. This allows the instance SIL callback order to be set in addition to the object priority.
- **Startup Hook:** The callback function that is invoked before any changes done to the <startup> datastore. It is invoked only if the <startup> capability is enabled.
- **Dynamic Default Hook callback:** The callback function that is the user/system callback that can be used to set up dynamic default value to the non-default leafy nodes during load or edit-config operations. It is invoked when the server checks if the node has any defaults and if there is no any YANG defined defaults the server can update the node with custom “system” default value.

The following API functions are supported that can be called from the **Set Hook** or **Post Set Hook** callbacks:

- **Add Edit** (agt_val_add_edit): Add an edit operation to the current transaction
- **Get Data** (agt_val_get_data): Retrieve an instance of a data node from the server.

The following API functions are supported that can be called from SIL-SA code for the **Set Hook** or **Post Set Hook** callbacks:

- **Add Edit** (sil_sa_add_edit): Add an edit operation to the current transaction
- **Get Data** (sil_sa_get_data): Retrieve an instance of a data node from the server.

The **netconfd-pro** provides the following SIL-SA support for In-Transaction APIs:

- **SA Transaction Start:** The callback that is intended to provide access to the transaction control block and its parameters right before the transaction is started. For SIL-SA usage the server does not provide transaction control block, instead it passes the transaction ID for reference and some crucial information about the transaction, such as the information about whether the current transaction is for the running datastore, is the transaction for validate operation, or if it is for Rollback Phase.
- **SA Transaction Complete:** The callback that is intended to provide access to the transaction control block and its parameters right after the transaction is completed. Called for every transaction completion, not just transactions that complete without errors. For SIL-SA usage the server does not provide transaction control block, instead it passes only the transaction ID for reference.

- **SA Set Hook:** The callback that is intended to alter the data in the current edit and or add new edits to the current transaction. For SIL-SA version of this callback the transaction control block is not available, instead the server provides all necessary information as a set of parameters.
- **SA Post Set Hook:** The callback that is intended to alter the data in the current edit and or add new edits to the current transaction after the EDIT callback for the same node. For SIL-SA version of this callback the transaction control block is not available, instead the server provides all necessary information as a set of parameters.
- **SA Transaction Hook:** The callback that is intended to provide validation point for specific node(s). For SIL-SA usage the server does not provide transaction control block, instead it passes the transaction ID for reference and some crucial information about the transaction, such as the information about whether the current transaction is for the running datastore, is the transaction for validate operation, or if it is for Rollback Phase.
- **SA Validate Complete:** The callback that is intended to provide validation point after the Validate Phase is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.
- **SA Apply Complete:** The callback that is intended to provide validation point after the Apply Phase is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.
- **SA Commit Complete:** The callback that is intended to provide validation point after the Commit Phase is completed. For SIL-SA usage the server passes the transaction ID for reference, commit type and **Get Data** API can be used to run additional validation if required.
- **SA Rollback Complete**: The callback that is intended to provide validation point after the Rollback Phase is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.

If an application needs to write more data in the datastore in the same transaction, there are mechanisms implemented in the form of **API functions** called **Set Hook** and **Post Set Hook.**

The **Set Hook and Post Set Hook** callbacks are the ways for the application to manipulate with the datastore in the same transaction. For example, whenever the specific node is created, the hook registered on that node gets called and the specific list can be updated with additional list entry or entries.

If an application needs to perform additional validations, security check or any other manipulations with datastore in the same transaction, there are mechanisms implemented in the form of **API functions** called **Transaction Hook, Start Transaction, and Complete Transaction**.

If an application needs to perform additional actions and manipulations with datastore in the same transaction during **<commit>** operation, there are mechanisms implemented in the form of **API functions** called **Startup Hook**, **Validate Complete, Apply Complete, Commit Complete or Rollback Complete** callbacks. These callbacks are called **Commit Completeness** callbacks.

Manipulation with configuration data are only allowed for **Set Hook and Post Set Hook** callbacks, however. If **Transaction Hook, Start/Complete Transaction, or Commit Completeness** callbacks are trying to adjust configuration data via **Add Edit** API (an API function that allows to add edits on the fly within the same transaction, described later), the operation will be skipped. However, this action will not generate an error, just a warning message in the log with log-level equals debug2.

NOTE:

In-Transaction callbacks are NOT part of the **yangdump-sdk** code generation. After the **make_sil_dir_pro** script is run, In-Transaction callbacks will NOT be auto-generated. All the In-Transaction callbacks are optional and merely intend to provide more efficient way to manipulate with the database and give an access to the database at the specific phase, transaction, and an edit.

7.7.1 Set-Hook Callback

A **Set Hook** is a function that is invoked within the transaction when an object is modified. When **netconfd-pro** server has been configured to provide a candidate configuration, **Set Hook** code will be invoked when changes are done to the **<candidate>** configuration if the **--target=candidate** parameter is used. If **--target=running** then the set hook will be invoked at the start of the transaction on the running datastore. This callback will be invoked before a EDIT-1 or EDIT2 callback for the same object. Refer to netconfd development manual for more information.

Set-Hook can be invoked during the Load; However, it is not allowed to add new edits. So, callback are treated as **Transaction-Hook** style hooks and can perform only validation tasks and cannot edit datastore. Any **Add Edit** API during Load will be ignored, it is not an error just skip the call and go back with NO_ERR status.

Callback Template

- Type: User Callback
- Max Callbacks: 1 set-hook callback per object
- File: **agt_cb.h**
- Template: **agt_cb_hook_t**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **txcb** == transaction control block in progress
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == container object holding the proposed changes to apply to the current config, depending on the editop value.
    - **curval** == current container values from the <running> or <candidate> configuration, if any. Could be NULL for create and other operations.
  - Outputs: none
  - Returns: **status_t:**
    Status of the callback function execution
- Register: **agt_cb_hook_register**
- Unregister: **agt_cb_hook_unregister**

```
/* Typedef of the agt_hook_cb callback */
typedef status_t
      (*agt_cb_hook_t)(ses_cb_t *scb,
                       rpc_msg_t *msg,
                       agt_cfg_transaction_t *txcb,
                       op_editop_t editop,
                       val_value_t  *newval,
                       val_value_t  *curval);
```

7.7.2 Set Hook Callback Examples

The **Set Hook** can be set to specific object at run-time with a callback as follows:

Example 1:

Register the Set hook to the "**/example"** node of 'yang' YANG module. The hook is setup in platform-specific way with an API function, without needing to alter the YANG file.

```
    /* Register an object specific Set Hook callback */
    agt_cb_hook_register((const xmlChar *)"/yang:example",
                         AGT_HOOK_FMT_NODE,
                         AGT_HOOK_TYPE_SETHOOK,
                         hooks_sethook_edit );
```

Now, whenever the node **"/example"** is edited, the callback function will be called and additional specific data can be updated with desired values.

The callback function could look like:

```
/*********************************************************************
* FUNCTION hooks_sethook_edit
*
* Callback function for server object handler
* Used to provide a callback for a specific named object
*
* Set-Hook:
*   trigger: editop /example
*   add_edit:
*       add nodes: update 1 list entry with key=15
*
*           path: /if:interfaces/interface[key=15]
*
*********************************************************************/
static status_t
    hooks_sethook_edit (ses_cb_t *scb,
                        rpc_msg_t *msg,
                        agt_cfg_transaction_t *txcb,
                        op_editop_t editop,
                        val_value_t  *newval,
                        val_value_t  *curval)
{
    log_debug("\nEnter SET-Hook callback");

    status_t res = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    const xmlChar *defpath =
        (const xmlChar *)"/if:interfaces";


    switch (editop) {
    case OP_EDITOP_LOAD:
        break;
    case OP_EDITOP_MERGE:
```

```
    case OP_EDITOP_REPLACE:
    case OP_EDITOP_CREATE:
        /* add a new edit if the "/yang:example" value is "example" */
        if (newval &&
            !xml_strcmp(VAL_STR(newval), (const xmlChar *)"example")) {

            /* find object template of the desired node */
            obj_template_t *targobj =
                      ncx_match_any_object_ex((const xmlChar *)"ietf-interfaces",
                                              (const xmlChar *)"interfaces",
                                              FALSE,
                                              NCX_MATCH_FIRST,
                                              FALSE,
                                              &res);
            if (!targobj) {
                return ERR_NCX_INVALID_VALUE;
            }

           /* create edit_value container value */
            val_value_t *editval = val_new_value();
            if (editval == NULL) {
                return ERR_INTERNAL_MEM;
            }
            val_init_from_template(editval, targobj);

            /* malloce and construct list value, for more
             * examples refer to libhooks-test/src/hooks-test.c library
             */
            uint32 key = 11;
            val_value_t *list_value = create_list_entry(VAL_OBJ(editval),
                                                        key,
                                                        &res);
            if (!list_value) {
                val_free_value(editval);
                return res;
            }

            /* add a new list entry */
            val_add_child_sorted(list_value, editval);

            /* add a new edit, MERGE on defpath with 1 new list entry */
            if (res == NO_ERR) {
                res = agt_val_add_edit(scb,
                                       msg,
                                       txcb,
                                       defpath,
                                       editval,
                                       OP_EDITOP_MERGE);
            }

            /* clean up the editval */
            val_free_value(editval);
        }
        break;
    case OP_EDITOP_DELETE:
        /* delete the interfaces container if the curval of 'example' node is
             "deleteall" */

       if (curval &&
            !xml_strcmp(VAL_STR(curval), (const xmlChar *)"delete-all")) {

            res = agt_val_add_edit(scb,
                                   msg,
```

```
                                    txcb,
                                    defpath,
                                    NULL, // editval
                                    editop);
        }
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_CONTENT,
            res,
            NULL,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval);
    }

    return res;

}  /* hooks_sethook_edit */
```

So whenever some north bound agent edit an "/**example"** node the callback kicks in and additionally creates a new interface "**/interfaces/interface[key=15]"**

Or, deletes the whole **/interfaces** container if "**/example"** value is set to 'delete-all' and some north bound agent attempts to DELETE "**/example"**.

Example 2:

Register the Set hook to the "**/example2"** container of 'yang' YANG module. The hook is setup in platform-specific way with an API function, without needing to alter the YANG file.

```
    /* Register an object specific Set Hook callback */
    res = agt_cb_hook_register((const xmlChar*)"/yang:topcontainer/yang:example2",
                               AGT_HOOKFMT_NODE,
                               AGT_HOOK_TYPE_SETHOOK,
                               hooks_sethook5_edit);
```

Now, when user wants to create a "/**example2"** container, the callback function will be called and container can be updated with desired additional values.

The callback function could look like:

```
/********************************************************************
* FUNCTION  hooks_sethook_edit2
*
```

```
* Callback function for server object handler
* Used to provide a callback for a specific named object
*
* Set-Hook:
*   trigger: create /topcontainer/example2
*       create container and populate extra nodes
*
*   add_edit effect: populate extra nodes within the child container
*        when /topcontainer/example2 is created
*
*  Note: callback format should be AGT_HOOKFMT_NODE in order to invoke
*   this callback only for a new container edit.
*   If the format is AGT_HOOKFMT_SUBTREE, the callback will be invoked
*   if you edit children as well. In this case newval will not
*   represent container value and cannot not be used to construct
*   a new editval.
*
*********************************************************************/
static status_t
    hooks_sethook_edit2 (ses_cb_t *scb,
                         rpc_msg_t *msg,
                         agt_cfg_transaction_t *txcb,
                         op_editop_t editop,
                         val_value_t  *newval,
                         val_value_t  *curval)
{
    log_debug("\nEnter SET-Hook callback");

    status_t res = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    /* defpath specified target root */
    const xmlChar *defpath =
        (const xmlChar *)"/yang:topcontainer/yang:example2";

    switch (editop) {
    case OP_EDITOP_LOAD:
        break;
    case OP_EDITOP_MERGE:
    case OP_EDITOP_REPLACE:
       break;
    case OP_EDITOP_CREATE:
        /* populate a new container with several leafs */
        if (newval) {

            /* use newval value to write a new edits */
            val_value_t *editval =
                val_clone_config_data(newval, &res);
            if (editval == NULL) {
                return ERR_INTERNAL_MEM;
            }

            /* malloced and construct some leaf values */
            val_value_t *child =
                agt_make_leaf(VAL_OBJ(editval),
                              (const xmlChar *)"exampleleaf",
                              (const xmlChar *)"leaf-value",
                               &res);
            if (!child) {
                val_free_value(editval);
                return ERR_NCX_INVALID_VALUE;
            }
            val_add_child(child, editval);
```

```
            /* add a new edit on defpath and populate new entries */
            if (res == NO_ERR) {
                res = agt_val_add_edit(scb,
                                       msg,
                                       txcb,
                                       defpath,
                                       editval,
                                       OP_EDITOP_MERGE);
            }

            /* clean up the editval */
            val_free_value(editval);
        }
        break;
    case OP_EDITOP_DELETE:

        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_CONTENT,
            res,
            NULL,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval);
    }

    return res;

}  /* hooks_sethook_edit2 */
```

So whenever some north bound agent creates an /**topcontainer/example2** container the callback kicks in and additionally creates a new leafs for this container

For more detailed examples refer to **libhooks-test/src/hooks-test.c** library.

7.7.3 SIL-SA Set-Hook Callback

The following sections describe how to use Set Hook within **SIL-SA** code and what are the differences between regular **Set Hook** usage within **SIL** code and **SIL-SA** code.

A **Set Hook** is a function that is invoked within the transaction when an object is modified. When the **netconfd-pro** server has been configured to provide a candidate configuration, **Set Hook** code will be invoked when changes are done to the <candidate> configuration. If **--target=running** then the **Set Hook** will be invoked at the start of the transaction on the running datastore.

In the **SIL-SA** there is no any Transaction control block **TXCB;** however, the callbacks still need to have sufficient substitution for all the crucial flags from Transaction control block. As a result the **SIL-SA** callback template is different.

The following function template definition is used for **Set Hook** callback for **SIL-SA** functions:

Callback Template

- Type: User Callback
- Max Callbacks: 1 Set Hook callback per object
- File: **agt_cb.h**
- Template: **agt_cb_sa_hook_t**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == value holding the proposed changes to apply to the current config, depending on the editop value
    - **curval** == current values from the <running> or <candidate> configuration, if any. Could be NULL for create and other operations
    - **transaction_id ==** transaction ID of the transaction control block in progress
    - **isvalidate** == TRUE if this Transaction is for <validate> operation
    - **isload ==** TRUE if this Transaction is for a Load operation
    - **isrunning ==** TRUE if this Transaction is for the the running datastore
  - Outputs: none
  - Returns: **status_t:**
    Status of the callback function execution
- Register: **agt_cb_sa_hook_register**
- Unregister: **agt_cb_sa_hook_unregister**

```
/* Typedef of the agt_cb_sa_hook_t callback */
typedef status_t
    (*agt_cb_sa_hook_t) (ses_cb_t *scb,
                         rpc_msg_t *msg,
                         op_editop_t editop,
                         val_value_t *newval,
                         val_value_t *curval,
                         const xmlChar *transaction_id,
                         boolean isvalidate,
                         boolean isload,
                         boolean isrunning);
```

7.7.4 SIL-SA Set Hook Callback Examples

The following sections illustrates how to utilize the **Set Hook** callback in examples.

NOTE:

Manipulation with datastore are only allowed for **Set Hook** callbacks. If **Transaction Hook** or other In-Transaction callbacks call **Add Edit** API the operation will be ignored. It will not return an error, just ignore **Add Edit** API calls.

Let us go through simple examples that will illustrate how to utilize the **Set Hook** callbacks for the specific purposes. First we need a YANG module. Consider this simplified, but functional, example. You can download this YANG module from attachments and run **make_sil_sa_dir** to auto-generate stub **SIL-SA** code for this module.

```
module silsa-sethook-example {
  namespace "http://netconfcentral.org/ns/silsa-sethook-example";
  prefix "sa-sethook-ex";

  revision 2020-08-18 {
    description "Initial revision.";
  }

  container interfaces {
    list interface {
      key "name";

      leaf name {
        type string;
      }
      leaf speed {
        type enumeration {
          enum 10m;
          enum 100m;
          enum auto;
        }
      }
      leaf hook-node {
        type uint32;
      }
      container state {
        leaf admin-state {
          type boolean;
        }
      }
    }

    leaf status {
      type string;
    }
  }

  leaf trigger {
    type string;
  }
}
```

Note, the **Set Hook** callback is not part of the auto-generated code and you will need to modify this stub **SIL-SA** code and add registration for your **Set Hook** callback functions.

SIL-SA code generation command that is used in this example.

```
      > make_sil_sa_dir silsa-sethook-example --sil-get2
```

Assume we registered **Set Hook** callback for the “trigger” leaf node. Thus, whenever the node **/trigger** is edited, the **Set Hook** callback function will be called and additional specific data can be updated or populated with desired values.

In this example, we will generate an extra “interface” list entry with key value equal to “vlan1” when a “trigger” node is getting edited with as specific value equal to “add-edit”. The callback function may look as follows:

```
/********************************************************************
* FUNCTION  sethook_callback
*
* Callback function for server object handler
* Used to provide a callback for a specific named object
*
* Set Hook:
*   trigger: edit /trigger
*   add_edit:
*       add nodes: populate 1 list entry with name=vlan1
*
*           path: /interfaces/interface[name=vlan1]
*
*********************************************************************/
static status_t
    sethook_callback (ses_cb_t *scb,
                      rpc_msg_t *msg,
                      op_editop_t editop,
                      val_value_t *newval,
                      val_value_t *curval,
                      const xmlChar *transaction_id,
                      boolean isvalidate,
                      boolean isload,
                      boolean isrunning)
{
    status_t res = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();

    if (LOGDEBUG2) {
        log_debug2("\n\n*******************************************");

        print_callback_info(errorval,
                            AGT_CB_VALIDATE,
                            editop,
                            (const xmlChar *)"SETHOOK");

        log_debug2("\ntransaction_id -- %s", transaction_id);
        log_debug2("\nuser_id -- %s", user);
        log_debug2("\nclient_addr -- %s", client_addr);
        log_debug2("\nisvalidate -- %s",
```

```
            isvalidate ? NCX_EL_TRUE : NCX_EL_FALSE);
        log_debug2("\nisload -- %s",
            isload ? NCX_EL_TRUE : NCX_EL_FALSE);
        log_debug2("\nisrunning -- %s",
            isrunning ? NCX_EL_TRUE : NCX_EL_FALSE);
        log_debug2("\n*****************************************\n\n");
    }

    const xmlChar *defpath =
        (const xmlChar *)"/sa-sethook-ex:interfaces";

    switch (editop) {
    case OP_EDITOP_LOAD:
        break;
    case OP_EDITOP_MERGE:
    case OP_EDITOP_REPLACE:
    case OP_EDITOP_CREATE:
        /* add a new edit if the "/trigger" value is "add-edit" */
        if (newval &&
            !xml_strcmp(VAL_STR(newval), (const xmlChar *)"add-edit")) {

            /* find object template of the desired node */
            obj_template_t *targobj =
                ncx_find_object(silsa_sethook_example_mod,
                                (const xmlChar *)"interfaces");
            if (!targobj) {
                return ERR_NCX_INVALID_VALUE;
            }

           /* create edit_value container value */
            val_value_t *editval = val_new_value();
            if (editval == NULL) {
                return ERR_INTERNAL_MEM;
            }
            val_init_from_template(editval, targobj);

            /* malloc and construct list value */
            val_value_t *list_value =
                  create_list_entry(VAL_OBJ(editval),
                                    (const xmlChar *)"vlan1",
                                    &res);
            if (!list_value) {
                val_free_value(editval);
                return res;
            }

            /* add a new list entry */
            res = val_child_add(list_value, editval);
            if (res != NO_ERR) {
                val_free_value(list_value);
            } else {
                /* add a new edit, MERGE on defpath with 1 new list entry */
                const xmlChar *edit_operation = (const xmlChar *)"merge";
                const xmlChar *insert_point = NULL;
                const xmlChar *insert_where = NULL;
                boolean skip_cb = FALSE;

                res =
                    sil_sa_add_edit(defpath,
                                    editval,
                                    edit_operation,
                                    insert_where,
                                    insert_point,
```

```
                                    skip_cb);
            }

            /* clean up the editval */
            val_free_value(editval);
        }
        break;
    case OP_EDITOP_DELETE:
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(scb,
                         &msg->mhdr,
                         NCX_LAYER_CONTENT,
                         res,
                         NULL,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval);
    }

    return res;

}  /* sethook_callback */
```

NOTE:

When you are constructing a list node make sure that a key node is getting created first, and then all other children are getting added. Also, make sure that you use **val_gen_index_chain** API function after you construct the list, so it will be normalized and all required fields will be set accordingly.

The edit-config that will trigger desired Set Hook actions may look as follows:

```
<edit-config>
  <target>
    <candidate/>
  </target>
  <default-operation>merge</default-operation>
  <test-option>set</test-option>
  <config>
    <trigger
      xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
      nc:operation="create"
      xmlns="http://netconfcentral.org/ns/silsa-sethook-example">add-edit</trigger>
  </config>
</edit-config>
```

As a result, whenever some north bound agent edit the **/trigger** with a specific value, the callback is invoked and additionally creates a new interface **/interfaces/interface[name=value]**.

To ensure that the data added by **subsystem** was successfully generated an application can retrieve configurations. The server should reply with:

```
<data>
 <interfaces xmlns="http://netconfcentral.org/ns/silsa-sethook-example">
  <interface>
   <name>vlan1</name>
   <hook-node>1000</hook-node>
  </interface>
 </interfaces>
 <trigger xmlns="http://netconfcentral.org/ns/silsa-sethook-example">add-edit</trigger>
</data>
```

For more detailed examples refer to **libhooks-test-silsa/src/hooks-test-silsa.c** library.

7.7.5 Post Set Hook Callback

A **Post Set Hook** callback - is a postponed **Set Hook** callback analogous function that is invoked within the transaction when an object is modified but **AFTER EDIT** callback is done for the same object. When the **netconfd-pro** server has been configured to provide a candidate configuration, **Post Set Hook** code will be invoked when changes are done to the **<candidate>** configuration if the **--target=candidate** parameter is used. If **--target=running** then the **Post Set Hoo**k will be invoked at the start of the transaction on the **<running>** datastore. This callback will be invoked **AFTER** a **EDIT1** or **EDIT2** callback for the same object.

NOTE:

This callback will not be invoked in case the **--sil-root-check-first** is set to TRUE when the server is run with **--target=running**. The server will ignore the callback and just skip its invocation.

Post Set Hook can be invoked during the Load; However, it is not allowed to add new edits. So, callbacks are treated as **Transaction Hook** style callbacks and can perform only validation tasks and cannot edit datastore. Any **Add Edit** API during Load will be ignored, it is not an error just skip the call and go back with NO_ERR status.

The following function template definition is used for **Post Set Hook** callback functions:

```
/* Typedef of the agt_hook_cb callback */
typedef status_t
      (*agt_cb_hook_t)(ses_cb_t *scb,
                       rpc_msg_t *msg,
                       agt_cfg_transaction_t *txcb,
                       op_editop_t editop,
                       val_value_t *newval,
                       val_value_t *curval);
```

Callback Template

- **Type: User Callback**
- Max Callbacks: 1 Post Set Hook callback per object
- File: **agt_cb.h**
- Template: **agt_cb_hook_t**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **txcb** == transaction control block in progress
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == value holding the proposed changes to apply to the current config, depending on the editop value.
    - **curval** == current values from the <running> or <candidate>configuration, if any. Could be NULL for create and other operations.
  - Outputs: none
  - Returns: **status_t:** Status of the callback function execution
- Register: **agt_cb_post_hook_register**
- Unregister**: agt_cb_post_hook_unregister**

7.7.6 SIL-SA Post Set Hook Callback

A SIL-SA version of the **Post Set Hook** is a postponed **Set Hook** callback analogous function that is invoked within the transaction when an object is modified but **AFTER EDIT** callback is done for the same object. When the **netconfd-pro** server has been configured to provide a candidate configuration, Post **Set Hook** code will be invoked when changes are done to the **<candidate>** configuration if the **--target=candidate** parameter is used. If --**target=running** then the Post Set Hook will be invoked at the start of the transaction on the **<running>** datastore. This callback will be invoked **AFTER EDIT** callback for the same object.

The following function template definition is used for **SIL-SA Post Set Hook** callback functions:

NOTE:

This callback will not be invoked in case the **--sil-root-check-first** is set to TRUE when the server is run with **--target=running**. The server will ignore the callback and just skip its invocation.

Post Set Hook can be invoked during the Load; However, it is not allowed to add new edits. So, callbacks are treated as **Transaction Hook** style callbacks and can perform only validation tasks and cannot edit datastore. Any **Add Edit API** during Load will be ignored, it is not an error just skip the call and go back with NO_ERR status.

The following function template definition is used for **Post Set Hook** callback functions:

```
/* Typedef of the agt_cb_sa_hook_t callback */
typedef status_t
    (*agt_cb_sa_hook_t) (ses_cb_t *scb,
                         rpc_msg_t *msg,
                         op_editop_t editop,
                         val_value_t *newval,
                         val_value_t *curval,
                         const xmlChar *transaction_id,
                         boolean isvalidate,
                         boolean isload,
                         boolean isrunning);
```

Callback Template

- Type: User Callback
- Max Callbacks: 1 Set Hook callback per object
- File: **agt_cb.h**
- Template: **agt_cb_sa_hook_t**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == value holding the proposed changes to apply to the current config, depending on the editop value
    - **curval** == current values from the <running> or <candidate> configuration, if any. Could be NULL for create and other operations
    - **transaction_id ==** transaction ID of the transaction control block in progress

        - **isvalidate** == TRUE if this Transaction is for <validate> operation
        - **isload ==** TRUE if this Transaction is for a Load operation
        - **isrunning ==** TRUE if this Transaction is for the the running datastore
    - Outputs: none
    - Returns: **status_t:**
      Status of the callback function execution
- Register: **agt_cb_sa_post_sethook_register**
- Unregister: **agt_cb_sa_post_sethook_unregister**

7.7.7 Transaction Hook Callback

A **Transaction Hook** is a function that is invoked within the transaction when an object is modified. The **Transaction Hook** is similar to the **Set-Hook** except this callback is invoked just before the data is committed to the running datastore. This callback will be invoked after EDIT-1 or EDIT2 callbacks for the same object. Note that the **Transaction-hook** is not allowed to change anything, It is not allowed to call **Add Edit** API, or to alter the edits or the datastore in any way. Manipulation with datastore are only allowed for **Set-Hook** callbacks. If **Transaction-Hook** callbacks calls **Add Edit** API, the operation will be ignored. Do not return error just ignore **Add Edit** API calls.

Callback Template

- Type: User Callback
- Max Callbacks: 1 transaction-hook callback per object
- File: **agt_cb.h**
- Template: **agt_cb_hook_t**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming rpc_msg_t in progress
    - **txcb** == transaction control block in progress
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == container object holding the proposed changes to apply to the current config, depending on the editop value. Will not be NULL.
    - **curval** == current container values from the <running> or <candidate> configuration, if any. Could be NULL for create and other operations.
  - Outputs: none
  - Returns: **status_t:**
    Status of the callback function execution
- Register: **agt_cb_hook_register**
- Unregister**:** **agt_cb_hook_unregister**

```
/* Typedef of the agt_hook_cb callback */
typedef status_t
      (*agt_cb_hook_t)(ses_cb_t *scb,
                       rpc_msg_t *msg,
                       agt_cfg_transaction_t *txcb,
                       op_editop_t editop,
                       val_value_t  *newval,
                       val_value_t  *curval);
```

7.7.8 Transaction Hook Callback Examples

The **Transaction Hook** can be set to specific object at run-time with a callback as follows:

Example 1:

Register the **Transaction Hook** to the **/example** node of 'yang' YANG module. The hook is setup in platform-specific way with an API function, without needing to alter the YANG file.

```
   /* Register an object specific Transaction Hook callback */
   agt_cb_hook_register((const xmlChar *)”/yang:example”,
                        AGT_HOOK_FMT_NODE,
                        AGT_HOOK_TYPE_TRANSACTION,
                        hooks_sethook_edit );
```

Now, whenever the 'example' node is edited, the callback function will be called and perform specific validation actions.

The callback function could look like:

```
/********************************************************************
* FUNCTION hooks_transhook_edit
*
* Callback function for server object handler
* Used to provide a callback for a specific named object
*
* Transaction-Hook:
*   trigger: DELETE /interface/interfaces[key]
*   effect:
*       - if testval node exist the DELETE operation will be denied
*       - if testval is false, the operation will be denied
*
* Manipulation with datastore are only allowed for Set-Hook
* callbacks. If Transaction Hook or Start/Complete
* Transaction callbacks call add_edit() the operation will be
* ignored. Do not return error just ignore add_edit calls.
*
*********************************************************************/
static status_t
    hooks_transhook_edit (ses_cb_t *scb,
                          rpc_msg_t *msg,
                          agt_cfg_transaction_t *txcb,
                          op_editop_t editop,
                          val_value_t  *newval,
                          val_value_t  *curval)
{
    log_debug("\nEnter Transaction-Hook callback");

    status_t res = NO_ERR;
    status_t res2 = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    const xmlChar *defpath =
      (const xmlChar *)"/if:interfaces/if:interface[if:name]/if:enabled";

    val_value_t *testval =
```

```
        agt_val_get_data(txcb->cfg_id,
                         defpath,
                         &res2);

    switch (editop) {
    case OP_EDITOP_LOAD:
        break;
    case OP_EDITOP_MERGE:
    case OP_EDITOP_REPLACE:
    case OP_EDITOP_CREATE:
        if (testval) {
            res = ERR_NCX_ACCESS_DENIED;
        }
        break;
    case OP_EDITOP_DELETE:
        if (testval && VAL_BOOL(testval)) {
            res = ERR_NCX_ACCESS_DENIED;
        } else {
            res2 = NO_ERR;
        }
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(
            scb,
            &msg->mhdr,
            NCX_LAYER_CONTENT,
            res,
            NULL,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval,
            (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
            errorval);
    }

    return res;

}  /* hooks_transhook_edit */
```

So whenever some north bound agent edit an /**example** node the callback kicks in and additionally validates the **/interfaces/interface[name]/enabled** node.

7.7.9 SIL-SA Transaction Hook Callback

A SIL-SA version of the **Transaction Hook** is a function that is invoked within the transaction when an object is modified. The **Transaction Hook** is similar to the **Set Hook** except this callback is invoked just before the data is committed to the running datastore. This callback will be invoked after **EDIT** callbacks for the same object. Note that the **Transaction hook** is not allowed to change anything, It is not allowed to call **Add Edit** API, or to alter the edits or the datastore in any way. Manipulation with datastore are only allowed for **Set Hook** callbacks. If **Transaction Hook** callbacks calls **Add Edit** API, the operation will be ignored. Do not return error just ignore **Add Edit** API calls.

The following function template definition is used for **Transaction Hook** callback functions:

Callback Template

- Type: User Callback
- Max Callbacks: 1 Set Hook callback per object
- File: **agt_cb.h**
- Template: **agt_cb_sa_hook_t**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == value holding the proposed changes to apply to the current config, depending on the editop value
    - **curval** == current values from the <running> or <candidate> configuration, if any. Could be NULL for create and other operations
    - **transaction_id ==** transaction ID of the transaction control block in progress
    - **isvalidate** == TRUE if this Transaction is for <validate> operation
    - **isload ==** TRUE if this Transaction is for a Load operation
    - **isrunning ==** TRUE if this Transaction is for the the running datastore
  - Outputs: none
  - Returns: **status_t:**
    Status of the callback function execution
- Register: **agt_cb_sa_hook_register**
- Unregister: **agt_cb_sa_hook_unregister**

```
/* Typedef of the agt_cb_sa_hook_t callback */
typedef status_t
    (*agt_cb_sa_hook_t) (ses_cb_t *scb,
                         rpc_msg_t *msg,
                         op_editop_t editop,
                         val_value_t *newval,
                         val_value_t *curval,
                         const xmlChar *transaction_id,
                         boolean isvalidate,
                         boolean isload,
                         boolean isrunning);
```

7.7.10 SIL-SA Transaction Hook Callback Examples

The following sections illustrates how to utilize the **SIL-SA Transaction Hook** callback in examples.

NOTE:

Manipulation with datastore are only allowed for **Set Hook** or **Post Set Hook** callbacks. If **Transaction Hook** or other In-Transaction callbacks call **Add Edit** API the operation will be ignored. It will not return an error, just ignore **Add Edit** API calls.

The **Transaction Hook** can be set to specific object at run-time with a callback as follows. Assume we have a leaf node "example" in the YANG module. Register the Transaction Hook to the **/example** node of the example YANG module.

Now, whenever the **/example** node is edited, the **Transaction Hook** callback will be invoked and it will perform specific validation actions. The callback function may look as follows:

```
/********************************************************************
* FUNCTION hook_cb
*
* Callback function for server object handler
* Used to provide a callback for a specific named object
*
* Transaction-Hook:
*   trigger: edit /example
*   effect:
*       - if testval node exist the DELETE operation will be denied
*       - if testval is false, the operation will be denied
*
* INPUTS:
*   scb == session control block making the request
*   msg == incoming rpc_msg_t in progress
*   editop == edit operation enumeration for the node being edited
*   newval == container object holding the proposed changes to
*           apply to the current config, depending on
*           the editop value. Will not be NULL.
*   curval == current container values from the <running>
*           or <candidate> configuration, if any. Could be NULL
*           for create and other operations.
*  transaction_id == transaction ID of the transaction control block in progress
*  isvalidate == TRUE if this Transaction is for <validate> operation
*  isload == TRUE if this Transaction is for a Load operation
*  isrunning == TRUE if this Transaction is for the the running datastore
*
* RETURNS:
*    status
*********************************************************************/
static status_t
    hook_cb (ses_cb_t *scb,
             rpc_msg_t *msg,
             op_editop_t editop,
             val_value_t *newval,
             val_value_t *curval,
```

```
             const xmlChar *transaction_id,
             boolean isvalidate,
             boolean isload,
             boolean isrunning)
{
    status_t res = NO_ERR;
    status_t res2 = NO_ERR;
    val_value_t *errorval = (curval) ? curval : newval;

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();

    if (LOGDEBUG2) {
        log_debug2("\n\n*****************************************");
        log_debug2("\nEnter hooks_sethook_edit callback for silsa-test "
                   "---- 1");
        log_debug2("\ntransaction_id -- %s", transaction_id);
        log_debug2("\nuser_id -- %s", user);
        log_debug2("\nclient_addr -- %s", client_addr);
        log_debug2("\nisvalidate -- %s",
            isvalidate ? NCX_EL_TRUE : NCX_EL_FALSE);
        log_debug2("\nisload -- %s",
            isload ? NCX_EL_TRUE : NCX_EL_FALSE);
        log_debug2("\nisrunning -- %s",
            isrunning ? NCX_EL_TRUE : NCX_EL_FALSE);
        log_debug2("\n*****************************************\n\n");
    }

    /* get the node that want to validate */
    val_value_t *testval =
        sil_sa_get_data(NCX_CFGID_RUNNING,
                        (const xmlChar *)"/if:interfaces/if:interface[if:name]/if:enabled",
                        &res2);

    switch (editop) {
    case OP_EDITOP_LOAD:
        break;
    case OP_EDITOP_MERGE:
    case OP_EDITOP_REPLACE:
    case OP_EDITOP_CREATE:
        if (testval) {
            res = ERR_NCX_ACCESS_DENIED;
        }
        break;
    case OP_EDITOP_DELETE:
        if (testval && VAL_BOOL(testval)) {
            res = ERR_NCX_ACCESS_DENIED;
        } else {
            res2 = NO_ERR;
        }
        break;
    default:
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }

    if (res != NO_ERR) {
        agt_record_error(scb,
                         &msg->mhdr,
                         NCX_LAYER_CONTENT,
                         res,
                         NULL,
                         (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                         errorval,
```

```
                          (errorval) ? NCX_NT_VAL : NCX_NT_NONE,
                          errorval);
    }

    return res;

}  /* hook_cb */
```

NOTE:

Manipulation with datastore are only allowed for **Set Hook** or **Post Set Hook** callbacks. If **Transaction Hook** callbacks call **Add Edit** API the operation will be ignored. It will not return an error, just ignore **Add Edit** API calls.

So whenever some north bound agent edit the **/example** node the callback is invoked and additionally validates the **/interfaces/interface[name]/enabled** node. Based on this validation, the operation can be denied.

7.7.11 Transaction Start Callback

The **Transaction Start** function is the user/system callback that is invoked before any changes to the database will be committed, right before the Validate phase and right after start of the Transaction.

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_trans_start_t**
  - Inputs:
    - **txcb** == transaction control block in progress
- Outputs: none
- Returns: **status_t:**

  Status of the callback function execution. If an error is returned the transaction will be terminated.
- Register: **agt_cb_trans_start_register**
- Unregister: **agt_cb_trans_start_unregister**

```
/* Typedef of the Transaction Start callback */
typedef status_t
    (*agt_cb_trans_start_t)(agt_cfg_transaction_t *txcb);
```

7.7.12 Transaction Start Callback Examples

The following example code illustrates how the **Transaction Start** callback may look like:

```
/********************************************************************
* FUNCTION transaction_start
*
* Start Transaction callback
* The Start Transaction function is the user/system
* callback that is invoked before any changes to the
* candidate database will be committed.
*
* Manipulation with datastore are only allowed for Set-Hook
* callbacks. If Transaction Hook or Start/Complete
* Transaction callbacks call add_edit() the operation will be
* ignored. Do not return error just ignore add_edit calls.
*
* Max Callbacks: No limit (except available heap memory)
* 1 Per SIL
*
* INPUTS:
*   txcb == transaction control block in progress
*
* RETURNS:
*   status
*********************************************************************/
static status_t
    transaction_start (agt_cfg_transaction_t *txcb)
{

    log_debug("\nEnter transaction_start callback");
    status_t res = NO_ERR;

    val_value_t *val =
       agt_val_get_data(txcb->cfg_id,
                        (const xmlChar *)"/yang:example",
                        &res);
    if (val) {
        res = ERR_NCX_ACCESS_DENIED;
    }

    return res;

}  /* transaction_start */
```

7.7.13 SIL-SA Transaction Start Callback

The SIL-SA version of the **Transaction Start** callback function is the sub-agent user/system callback that is invoked right before the transaction is started.

The server sends “server request” message to SIL-SA subsystems that registered callbacks right before the transaction is started. Once the SIL-SA subsystems receive the “server-message” they will invoke the callbacks. After the callbacks are invoked and if they are successful then subsystem responds with the “ok” message. If the callbacks are not invoked due to some reason or if the callbacks return error then an “error” message is returned to the server causing the server transaction termination.

The following function template definition is used for **Transaction Start** callback functions:

```
/* Typedef of the trans_start callback */
typedef status_t
    (*agt_cb_sa_trans_start_t) (const xmlChar *transaction_id,
                                boolean isvalidate,
                                boolean isrollback,
                                boolean isrunning);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_sa_trans_start_t**
  - Inputs:
    - **transaction_id** == transaction ID of the transaction control block in progress
    - **isvalidate == TRUE** if this is Transaction is for Validate operation
    - **isrollback == TRUE** if this is Transaction for a Rollback Phase or Load
    - **isrunning ==** if transaction is for the running datastore
- Outputs: none
- Returns: **status_t:**

  Status of the callback function execution. If an error is returned the transaction will be terminated.
- Register: **agt_cb_sa_trans_start_register**
- Unregister: **agt_cb_sa_trans_start_unregister**

7.7.14 SIL-SA Transaction Start Callback Examples

The following sections illustrates how to utilize the **Transaction Start** callback in examples.

In this example, the **Transaction Start** callback function applies multiple actions prior the transaction is actually started. The purpose of this function is to provide more validation options and more flexible and easier development. This callback function may deny the start of the transaction if some specific check is unmet.

In this example, if the transaction is for the <validate> operation on <running> datastore then the SIL-SA callback will trigger an error and deny the server transaction. In addition, for example, the callback function can send a custom notifications, or write a system or other log entries.

The following example code illustrates how the **Transaction Start** callback may look like.

```
/*****************************************************************
* FUNCTION silsa_transaction_start
*
* Start Transaction SA callback
* The Start Transaction function is the user/system
* callback that is invoked before any changes to the
* database will be committed.
*
* RETURNS:
*   status
*****************************************************************/
static status_t
    silsa_transaction_start (const xmlChar *transaction_id_val,
                             boolean isvalidate_val,
                             boolean isrollback_val,
                             boolean isrunning_val)
{
    if (!transaction_id_val) {
        log_error("\ntransaction_id value not set");
        return ERR_INTERNAL_VAL;
    }

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();

    if (LOGDEBUG2) {
        log_debug2("\n\n*****************************************"
                   "\nEnter silsa_transaction_start callback for silsa-test "
                   "---- 2"
                   "\ntransaction_id -- %s"
                   "\nuser_id -- %s"
                   "\nclient_addr -- %s"
                   "\nisvalidate -- %s"
                   "\nisrollback -- %s"
                   "\nisrunning -- %s"
                   "\n*****************************************\n\n",
                   transaction_id_val,
                   user,
                   client_addr,
                   isvalidate_val ? NCX_EL_TRUE : NCX_EL_FALSE,
                   isrollback_val ? NCX_EL_TRUE : NCX_EL_FALSE,
                   isrunning_val ? NCX_EL_TRUE : NCX_EL_FALSE);
    }

    /* return an error when "validate" operation is in progress */
    if (!isrollback_val && isvalidate_val && isrunning_val) {
```

```
        return ERR_NCX_INVALID_NUM;
    } else {
        return NO_ERR;
    }

}  /* silsa_transaction_start */
```

In the example above, the callback simply logs all the available parameters and denies the transaction if it is for <validate> operation on running datastore and return “Invalid Number” error status, as an example.

In this callback, there could be handling for customer specific pointers, it is a good place to initialize specific pointers that will be cleaned after the transaction complete during the **Transaction Complete** callback invocation.

In the example, we used two additional APIs to retrieve current transaction “user name” and “address”:

```
      /* Get the user_id value from the message header */
      const xmlChar *user = sil_sa_get_username();

      /* Get the client address (client_addr value from the message header) */
      const xmlChar *client_addr = sil_sa_get_client_addr();
```

These two APIs only available in the SIL-SA version of the SIL code and intended to provide an access to specific fields in the message header since the message header itself is not available in the SIL-SA version of of the SIL.

7.7.15 Transaction Complete Callback

The **Transaction Complete** function is the user/system callback that is invoked after the transactions has been processed.

The following function template definition is used for **Transaction Complete** callback functions:

```
/* Typedef of the Transaction complete callback */
typedef void
    (*agt_cb_trans_complete_t)(agt_cfg_transaction_t *txcb);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_trans_start_t**
  - Inputs:
    - **txcb** == transaction control block to check
- Outputs: none
- Returns: none

  Status of the callback function execution
- Register: **agt_cb_trans_complete_register**
- Unregister: **agt_cb_trans_complete_unregister**

7.7.16 Transaction Complete Callback Examples

The following example code illustrates how the **Transaction Complete** callback may look like:

```
/********************************************************************
* FUNCTION transaction_complete
*
* Complete Transaction callback
* The Transaction Complete function is the
* user/system callback that is invoked after
* the transactions has been processed.
*
* Max Callbacks: No limit (except available heap memory)
* 1 Per SIL
*
* INPUTS:
*   txcb == transaction control block in progress
*
* RETURNS:
*   none
*********************************************************************/
static void
    transaction_complete (agt_cfg_transaction_t *txcb)
{

    log_debug("\nEnter transaction_complete callback");

      /*   */


    return;

}  /* hooks_transaction_complete */
```

7.7.17 SIL-SA Transaction Complete Callback

The SIL-SA version of the **Transaction Complete** callback function is the user/system callback that is invoked after the transactions has been processed.

After the transaction is completed the server will send “server-event” to notify subsystems that the **Transaction Complete** callbacks should be invoked. During this process the server checks if a subsystem has any callbacks to be invoked and sends a “server-event” message to the subsystem if it has registered **Transaction Complete** callback(s). Once the SIL-SA subsystem receives the “server-event” for **Transaction Complete** callbacks it will invoke the callbacks.

The server does not expect that SIL-SA subsystems reply to the “server-event” message.

The following function template definition is used for **Transaction Complete** callback functions:

```
/* Typedef of the trans_start SA callback */
typedef status_t
    (*agt_cb_sa_trans_start_t) (const xmlChar *transaction_id);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_sa_trans_complete_t**
  - Inputs:
    - **transaction_id** == transaction ID of the transaction control block in progress
- Outputs: none
- Returns: none
- Register: **agt_cb_trans_complete_register**
- Unregister: **agt_cb_sa_trans_complete_unregister**

7.7.18 SIL-SA Transaction Complete Callback Examples

The following sections illustrates how to utilize the **Transaction Complete** callback in examples.

In this example, the **Transaction Complete** callback function applies multiple actions after the transaction is completed. The purpose of this function is to provide more validation options and more flexible and easier development. This callback function may send a custom notifications, or write a system or other log entries.

The following example code illustrates how the **Transaction Complete** callback may look like.

```
/********************************************************************
* FUNCTION transaction_complete
*
* Complete Transaction callback
* The Transaction Complete function is the
* user/system callback that is invoked after
* the transactions has been processed.
*
* INPUTS:
*   txcb == transaction control block in progress
*
* RETURNS:
*   status
********************************************************************/
static status_t
    transaction_complete (agt_cfg_transaction_t *txcb)
{

    log_debug("\nEnter transaction_complete callback");

    /* send custom notifications */

    /* write a sys, audit, vendor specific, etc log entries */


    return res;

}  /* transaction_complete */
```

7.7.19 Set Order Hook Callback

Set Order Hook callback Invoked in document order for each edited instance of the specified object. The callback returns the desired secondary SIL priority for the specific list instance . This callback is invoked once per edited instance and after any set-hook is called for the object and instance.

The following function template definition is used for **Set Order Hook** callback functions:

```
/* Typedef of the callback */
typedef uint8
    (*agt_cb_order_hook_t) (agt_cfg_transaction_t *txcb,
                            op_editop_t editop,
                            val_value_t  *newval,
                            val_value_t  *curval,
                            status_t *res);
```

Callback Template

- Type: User Callback
- Max Callbacks: 1 per object
- File: **agt_cb.h**
- Template: **agt_cb_order_hook_t**
  - Inputs:
    - **txcb** == transaction control block to check
    - **editop** == edit operation enumeration for the node being edited
    - **newval** == container object holding the proposed changes to apply to the current config, depending on the editop value.
    - **curval** == current container values from the <running> or <candidate> configuration, if any. Could be NULL for create and other operations.
    - **res** == address of return status
- Outputs: status of callback; status == error will cause the transaction to be terminated and rollback started
- Returns: the secondary SIL priority to assign to the object instance
- Register: **agt_cb_order_hook_register**
- Unregister: **agt_cb_order_hook_unregister**

7.7.20 Set Order Hook Callback Examples

The following sections illustrates how to utilize the **Set Order Hook** callback in examples.

The **Set Order Hook** callback function is the user/system callback that is used to provide an access for a specific list instance object and modify its secondary SIL priority.

NOTE:

The **Set Order Hook** callback function can be registered only for “list” YANG objects.

In this example, the **Set Order Hook** callback function applies priority to a list edit based on a key value. The purpose of this function is to prioritize edits and give more flexible and easier development.

The following example code illustrates how the **Set Order Hook** callback may look like:

```
/********************************************************************
* FUNCTION set_order_hook
*
* Callback function is
* used to provide a callback for a specific named object
*
* This callback is invoked once per instance before any
* Set Hook or Post Set Hook is called for the object and instance.
*
* INPUTS:
*   txcb == transaction control block in progress
*   editop == edit operation enumeration for the node being edited
*   newval == container object holding the proposed changes to
*           apply to the current config, depending on
*           the editop value. Will not be NULL.
*   curval == current container values from the <running>
*           or <candidate> configuration, if any. Could be NULL
*           for create and other operations.
*   res == address of return status
* OUTPUTS:
*    *res == status of callback; status == error will cause the
*            transaction to be terminated and rollback started
* RETURNS:
*    the secondary SIL priority to assign to the object instance
*********************************************************************/
static uint8
    set_order_hook (agt_cfg_transaction_t *txcb,
                    op_editop_t editop,
                    val_value_t  *newval,
                    val_value_t  *curval,
                    status_t *res)
{
    (void)txcb;
    (void)editop;
    (void)curval;

    uint8 retprior = 0;

    /* Set the priority only if the operation is not delete and if
```

```
     * a new value has keys, that will be compared later
     */
    if (newval && val_has_index(newval)) {

        /* Get the first index entry, if any for this value node */
       const val_index_t *c1 = val_get_first_index(newval);
       if (!c1) {
             return ERR_INTERNAL_VAL;
        }

        if (!xml_strcmp(VAL_STR(c1->val), (const xmlChar *)"vlan1")) {
            log_debug("\n Setting Priority to 140 for index:%s \n",
                VAL_STR(c1->val));

            retprior = 100;
        } else if (!xml_strcmp(VAL_STR(c1->val), (const xmlChar *)"ethernet1/1/10")) {
            log_debug("\n Setting Priority to 160 for index:%s \n",
                VAL_STR(c1->val));

            retprior = 150;
        } else {
            log_debug("\n Setting Priority to 130 for index:%s \n",
                VAL_STR(c1->val));

            retprior = 200;
        }
    }

    return retprior;

}  /* set_order_hook */
```

So, whenever some north bound agent edits an /**if:interfaces/if:interface** node the callback is invoked and additionally assigns desired priority to this edit. Note, that in the example above, the priority will be assigned only if there is a new value. It will be assigned only if the edit operation is not “delete” or “remove” because during the edits for those operations the server does not allocate a new value. Use a current value for any comparisons or validations during “delete” or “remove” edit operations.

To exemplify, if the north bound agent is creating 3 interfaces with different key values at the same time using the following <**edit-config**> RPC:

```
 <edit-config>
   <target>
     <candidate/>
   </target>
   <default-operation>merge</default-operation>
   <test-option>set</test-option>
       <config>
          <interfaces>
                  <interface>
                        <name>ethernet1/1/1</name>
                        </interface>

                  <interface>
                        <name>vlan1</name>
                        </interface>

                  <interface>
```

```
                        <name>ethernet1/1/10</name>
                    </interface>
          </interfaces>
        </config>
  </edit-config>
```

By the regular server logic, the server would validate/apply/commit these interfaces in order they are specified in the edit. However, using the **Set Order Hook** the server now will apply these “interfaces” based on their priority assigned in the callback function. The first list instance that will be processed by the server will be the “interface” with the key value set to “vlan1” since it has the highest priority - “100”.

So, the edits order would look as follows, from the server's logging:

```
***** start commit phase on candidate for session 5, transaction 1234358 *****
Start full commit of transaction 1234358: 3 edits on candidate config
edit-transaction 1234358: on session 5 by --@::1
  message-id: --
  trace-id: --
  datastore: candidate
  operation: create
  target: /if:interfaces/if:interface[if:name="vlan1"]
  comment: none

edit-transaction 1234358: on session 5 by --@::1
  message-id: --
  trace-id: --
  datastore: candidate
  operation: create
  target: /if:interfaces/if:interface[if:name="ethernet1/1/10"]
  comment: none

edit-transaction 1234358: on session 5 by --@::1
  message-id: --
  trace-id: --
  datastore: candidate
  operation: create
  target: /if:interfaces/if:interface[if:name="ethernet1/1/1"]
  comment: none
```

The **Set Order Hook** callback can be extremely useful in cooperation with the **Set Hook** or **Post Set Hook** callback or by itself. The power to prioritize all the edits in the transaction provides wide spectrum of utilization. The fact that the **Set Order Hook** is getting invoked before SIL or any other callbacks for the same object provides possibility to customize the transaction as desired.

7.7.21 Add Edit API

The **Add Edit** API is used to add an edit to the transaction in progress.

Manipulation with datastore are only allowed for **Set Hook** or **Post Set Hook** callbacks. If **Transaction Hook** or **Start/Complete Transaction** callbacks call **Add Edit** API the operation will be ignored. Do not return error just ignore **Add Edit** API calls.

The following function template definition is used for **Add Edit API** callback functions:

```
/* FUNCTION agt_val_add_edit */
status_t
       agt_val_add_edit(ses_cb_t *scb,
                        rpc_msg_t *msg,
                        agt_cfg_transaction_t *txcb,
                        const xmlChar *defpath,
                        val_value_t *edit_value,
                        op_editop_t editop);
```

Callback Template

- Type: Server Utility Function
- File: **agt_val.h**
- Template: **agt_val_add_edit**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **txcb** == transaction control block in progress
    - **defpath** == XPath expression specifying the data instance to add
    - **edit_value** == string containing the XML or JSON value to use in the edit. Can be NULL if editop is OP_EDITOP_DELETE or OP_EDITOP_REMOVE
    - **editop** == edit operation enumeration for the edit being added
  - Outputs: none
  - Returns: **status_t:**
    Status of the operation;
    - ERR_INTERNAL_MEM can be returned if a malloc fails
    - ERR_NCX_DEF_NOT_FOUND can be returned if the XPath expression contains unknown or ambiguous object names

7.7.22 Add Edit Extended API

The **Add Edit Extended** API is used to insert or move new list or leaf-list entries to the transaction in progress.

Manipulation with datastore are only allowed for **Set Hook** or **Post Set Hook** callbacks. If **Transaction Hook** or **Start/Complete Transaction** callbacks call **Add Edit** API the operation will be ignored. Do not return error just ignore **Add Edit** API calls.

The following function template definition is used for **Add Edit Extended API** callback functions:

```
/* FUNCTION agt_val_add_edit_ex */
status_t
    agt_val_add_edit_ex (ses_cb_t *scb,
                         rpc_msg_t *msg,
                         agt_cfg_transaction_t *txcb,
                         const xmlChar *defpath,
                         val_value_t *edit_value,
                         const xmlChar *edit_operation,
                         const xmlChar *insert_where,
                         const xmlChar *insert_point)
```

API Template

- Type: Server Utility Function
- File: **agt_val.h**
- Template: **agt_val_add_edit**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **txcb** == transaction control block in progress
    - **defpath** == XPath expression specifying the data instance to add
    - **edit_value** == val_value_t representing newnode in transaction only needed for create, merge, replace, insert. Ignored for delete or remove
    - **edit_operation** == <operation string>
      E.g.: "create" "delete" "insert" "merge" "move" "replace" "remove"
    - **insert_where** == <insert enum string>. Will be used only if operations are "move" or "insert". Ignored otherwise.
      E.g.: "before" "after" "first" "last"
    - **insert_point** == is a XPath encoded string like the defpath. Only for “before” or “after” **insert_where** parameter. The **insert_where** must be set to “before” or “after” if **insert_point** specified. Will be used only if the operations are "move" or "insert". Ignored otherwise.
      E.g: "/test3[string.1='entry2'][uint32.1='2']"
  - Outputs: none
  - Returns: **status_t:**
    Status of the operation;

NOTE:

The server does not support the "**insert**" and "**move**" operation on the nodes that are being modified at the same transaction at the same time. That's if the server already has an undo record (an edit on specific node), then the "**insert**" or "**move**" **Add Edit** API API on the same specific node will return an error. The server will only insert a new nodes and move only existent nodes. It will not move a new nodes or a nodes that are being already modified.

7.7.23 Add Edit Maximum API

The **Add Edit maximum** API is used to insert or move new list or leaf-list entries to the transaction in progress and also controls whether the server should invoke callbacks for added edits or not.
Manipulation with datastore are only allowed for **Set Hook** or **Post Set Hook** callbacks. If **Transaction Hook** or **Start/Complete Transaction** callbacks call **Add Edit** API the operation will be ignored. Do not return error just ignore **Add Edit** API calls.

The following function template definition is used for **Add Edit Maximum API** callback functions:

```
/* FUNCTION agt_val_add_edit_max */
status_t
    agt_val_add_edit_max (ses_cb_t *scb,
                          rpc_msg_t *msg,
                          agt_cfg_transaction_t *txcb,
                          const xmlChar *defpath,
                          val_value_t *edit_value,
                          const xmlChar *edit_operation,
                          const xmlChar *insert_where,
                          const xmlChar *insert_point,
                          boolean skip_cb);
```

API Template

- Type: Server Utility Function
- File: **agt_val.h**
- Template: **agt_val_add_edit_max**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming **rpc_msg_t** in progress
    - **txcb** == transaction control block in progress
    - **defpath** == XPath expression specifying the data instance to add
    - **edit_value** == val_value_t representing newnode in transaction only needed for create, merge, replace, insert. Ignored for delete or remove
    - **edit_operation** == <operation string>
      E.g.: "create" "delete" "insert" "merge" "move" "replace" "remove"
    - **insert_where** == <insert enum string>. Will be used only if operations are "move" or "insert". Ignored otherwise.
      E.g.: "before" "after" "first" "last"
    - **insert_point** == is a XPath encoded string like the defpath. Only for “before” or “after” **insert_where** parameter. The **insert_where** must be set to “before” or “after” if **insert_point** specified. Will be used only if the operations are "move" or "insert". Ignored otherwise.
      E.g: "/test3[string.1='entry2'][uint32.1='2']"
    - **skip_cb** ==TRUE if DO NOT invoke callbacks for an edded edit if any. FALSE if SKIP any callback for added edit including Transaction, EDIT1, EDIT2 callbacks. Only when **--target=running**.
  - Outputs: none
  - Returns: **status_t:**
    Status of the operation;

NOTE:

The server does not support the "**insert**" and "**move**" operation on the nodes that are being modified at the same transaction at the same time. That's if the server already has an undo record (an edit on specific node), then the "**insert**" or "**move**" **Add Edit** API on the same specific node will return an error. The server will only insert a new nodes and move only existent nodes. It will not move a new nodes or a nodes that are being already modified.

NOTE:

Skip_cb parameter can be used only when the default target <running>. Otherwise; the server always invoke callback for added edits in case the default target is <**candidate**>.

7.7.24 SIL-SA Add Edit API

The **SIL-SA** version of the **Add Edit** API has the same goal as regular **SIL** version; however, it has different template definition since the transaction control block is not available in the **SIL-SA** code. The whole functionality of **Add Edit** API is still available and all the edits will be added the same way as in **SIL** code.

The **Add Edit** API is used to insert or move new list or leaf-list entries to the transaction in progress and also controls whether the server should invoke callbacks for added edits or not.

Manipulation with datastore are only allowed for **Set Hook** or **Post Set Hook** callbacks. If **Transaction Hook** or **Start/Complete Transaction** callbacks call **Add Edit API** the operation will be ignored.

The following function template definition is used for **SIL-SA** version of the **Add Edit API** callback functions:

```
/* FUNCTION sil_sa_add_edit */
extern status_t
    sil_sa_add_edit (const xmlChar *defpath,
                     val_value_t *edit_value,
                     const xmlChar *edit_operation,
                     const xmlChar *insert_where,
                     const xmlChar *insert_point,
                     boolean skip_cb)
```

API Template

- Type: Server Utility Function
- File: **sil_sa.h**
- Template: **sil_sa_add_edit**
  - Inputs:
    - **defpath** == XPath path of object instance
    - **edit_value** == val_value_t representing newnode in transaction only needed for create, merge, replace, insert. Ignored for delete or remove
    - **edit_operation** == <operation string>
      E.g.: "create" "delete" "insert" "merge" "move" "replace" "remove"
    - **insert_where** == <insert enum string>. Will be used only if operations are "move" or "insert". Ignored otherwise.
      E.g.: "before" "after" "first" "last"
    - **insert_point** == is a XPath encoded string like the defpath. Only for “before” or “after” **insert_where** parameter. The **insert_where** must be set to “before” or “after” if **insert_point** specified. Will be used only if the operations are "move" or "insert". Ignored otherwise.
      E.g: "/test3[string.1='entry2'][uint32.1='2']"
    - **skip_cb** ==TRUE if DO NOT invoke callbacks for an added edit if any. FALSE if SKIP any callback for added edit including Transaction, EDIT callbacks.
  - Outputs: none
  - Returns: **status_t:**
    Status of the operation;

NOTE:

The server does not support the "**insert**" and "**move**" operation on the nodes that are being modified at the same transaction at the same time. That's if the server already has an undo record (an edit on specific node), then the "**insert**" or "**move**" **Add Edit** API on the same specific node will return an error. The server will only insert a new nodes and move only existent nodes. It will not move a new nodes or a nodes that are being already modified.

7.7.25 Get Data API

The **Get Data** API is used to retrieve data nodes from the server. This function can be used to access configuration data nodes. The data should not be altered. This API should be considered read-only.

If the XPath expression matches multiple nodes, then only the first instance is returned. The exact instance is implementation-dependent if the data is “ordered-by system”.

The following function template definition is used for **Get Data API** callback functions:

```
/* FUNCTION agt_val_get_data */
const val_value_t *
      agt_val_get_data (ncx_cfg_t  cfgid,
                        const xmlChar *defpath,
                        status_t *retres);
```

Callback Template

- Type: Server Utility Function
- File: **agt_val.h**
- Template: **agt_val_get_data**
  - Inputs:
    - **cfgid ==** datastore enumeration to use (candidate or running)
    - **defpath** == XPath expression specifying the data instance to add
    - **retres** == address of return status
  - Outputs:
    - ***retres** == set to the return status
  - Returns: **const *val_value_t**
    - Read only pointer to the requested data
    - Data cannot be saved and used later, can only be used immediately after the call to agt_val_get_data
    - ERR_INTERNAL_MEM can be returned if a malloc fails
    - ERR_NCX_DEF_NOT_FOUND can be returned if the XPath expression contains unknown or ambiguous object names

7.7.26 SIL-SA Get Data API

The **SIL-SA** version of the **Get Data** API has the same goal as regular **SIL** version; however, it has different template definition since the **SIL-SA** does not have a direct access to the server and the call to **Get Data** API will trigger message exchange with the server in order to retrieve the requested node.

If the XPath expression matches multiple nodes, then only the first instance is returned. The exact instance is implementation-dependent if the data is “ordered-by system”.

The following function template definition is used for **SIL-SA** version of the **Get Data API** callback functions:

```
/* FUNCTION sil_sa_get_data */
val_value_t *
    sil_sa_get_data (ncx_cfg_t cfg_id,
                     const xmlChar *defpath,
                     status_t *retres)
```

API Template

- Type: Server Utility Function
- File: **sil_sa.h**
- Template: **sil_sa_get_data**
  - Inputs:
    - **cfgid ==** datastore enumeration to use (candidate or running)
    - **defpath** == XPath expression specifying the data instance to add
    - **retres** == address of return status
  - Outputs:
    - ***retres** == set to the return status
  - Returns: **val_value_t**

This function will return value only if there is existing node in the datastore or there is defaults for the node. Data cannot be saved and used later, can only be used immediately after the call to **Get Data API.**

If the XPath expression "defpath" matches multiple nodes, then only the first instance is returned.
ERR_NCX_DEF_NOT_FOUND can be returned if the XPath expression contains unknown or ambiguous object names.

7.7.27 Startup Hook Callback

The **Startup Hook** function is the user/system callback that is invoked before any changes done to the <**startup**> datastore. It is invoked only if the <**startup**> capability is enabled.

The **Startup Hook** callback is invoked before any modifications to the <**startup**> datastore during <**edit-config**>, <**copy-config**> and <**delete-config**> operation. This callback is not limited hence any number of callbacks can be registered.

If an application needs to perform additional validations, check or any other actions before the <**startup**> datastore is modified or accessed, this new **Startup Hook** can be used.

The callback will be called for the following operations:

- any <**edit-config**> if the **--target=running** and <**startup**> is enabled
- After <**commit**> operation if --**target=candidate** and <**startup**> is enabled
- After <**copy-config**> if the target of the copying is the <**startup**> datastore
- <**delete-config**> if the target of deletion is <**startup**> datastore and the <**startup**> datastore is enabled.

Below CLI parameters can effect the callback invocation:

- **--with-startup=true** : <**copy-config**> and <**delete-config**> will invoke the callbacks
- **--with-startup=false** : <**copy-config**> and <**delete-config**> will NOT invoke the callbacks.

If the callback fails then no further actions will be executed and the server would reply with the error.

The following function template definition is used for **Startup Hook** callback functions:

```
/* Typedef of the startup_hook callback */
typedef status_t
      (*agt_cb_startup_hook_t)(ses_cb_t *scb,
                               rpc_msg_t *msg,
                               cfg_template_t source_config,
                               cfg_template_t target_config);
```

Callback Template:

- Type: User Callback
- Max Callbacks: No limit
- File: **agt_cb.h**
- Template: **agt_cb_startup_hook**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming rpc_msg_t in progress
    - **source** == datastore which is being copied
    - **target** == datastore which is being edited
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_startup_hook_register**

- Unregister**: agt_cb_startup_hook_unregister**

7.7.28 Startup Hook Callback Example

The following sections illustrates how to utilize the **Startup Hook** callback in examples.

The **Startup Hook** callback is called just before the **<startup>** is being modified.

The purpose of this function is to give more flexible and easier development for users. In case there is a need to access the **<startup>** datastore right before the server copies into it, there is a **Startup Hook** callback.

The following example code illustrates how the **Startup Hook** callback may look like:

```
/********************************************************************
* FUNCTION startup_hook_callback
*
* Startup Hook callback
* The Startup Hook Complete function is the
* user/system callback that is invoked before
* any changes to the <startup> during edit-config, copy-config and
* delete-config operation.
*
* Max Callbacks: No limit (except available heap memory)
*
* INPUT:
* scb == session control block making the request
* msg == incoming rpc_msg_t in progress
* source_config == datastore which is being copied
* target_config == datastore that is being edited
*
* RETURNS:
* status
*********************************************************************/
static status_t startup_hook_callback (ses_cb_t *scb,
                                       rpc_msg_t *msg,
                                       cfg_template_t *source_config,
                                       cfg_template_t *target_config)
{

      (void)scb;
      (void)msg;
      (void)source_config;
      (void)target_config;

      log_debug("\n\n Enter startup_hook_callback callback");

      /* notify application that the <startup> is being modified */
      log_debug("\n <startup> is being modified");

      /* write a custom logging */

      /* Run an extra validation, etc */

      return NO_ERR;

} /* startup_hook_callback */
```

7.7.29 Validate Complete Callback

The **Validate Complete** function is the user/system callback that is invoked after the **Validate Phase** has been processed during the <**commit**> operation.

If the callback fails the status of the failing callback is returned immediately and no further callbacks are made. As a result, the server will abort the commit.

The callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback is intended to allow manipulations with the running and candidate configurations (equivalent to completion of validation phase during <**commit**>).

NOTE:

The <**commit**> operation is not available if the default target is set to <**running**>, if the server is run with active **:writable:running** capability.

The callback is only called after commit operation for the specific phase has finished not after the validate for the specific module or edit is done.

The following function template definition is used for **Validate Complete** callback functions:

```
/* Typedef of the validate_complete callback */
typedef status_t
      (*agt_cb_validate_complete_t)(ses_cb_t *scb,
                                    rpc_msg_t *msg,
                                    val_value_t *candidate,
                                    val_value_t *running);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_validate_complete**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming rpc_msg_t in progress
    - **candidate** == candidate val_value_t for the config database to use
    - **running** == running val_value_t for the config database to use
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_validate_complete_register**

- Unregister: **agt_cb_validate_complete_unregister**

7.7.30 Validate Complete Callback Examples

The **Validate Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

All the registered callbacks will be called one after another right after the **Validate Phase** during the commit operation is done.

The callback is setup in platform-specific way with an API function. To register callback:

```
  /* Register Validate Complete callback */
   res = agt_cb_validate_complete_register(callback_function);
```

Now, whenever the validation phase is done for the <**commit**> operation, the callback function will be called and provide candidate and running datastores.

The callback function could look as follows:

```
/********************************************************************
* FUNCTION  callback_function
*
* Validate Complete callback
* The Validate Complete function is the
* user/system callback that is invoked after
* validation phase has been processed during <commit> op.
*
* Max Callbacks: No limit (except available heap memory)
*
* INPUTS:
*   scb == session control block making the request
*   msg == incoming rpc_msg_t in progress
*   candidate == candidate val_value_t for the config database to use
*   running == running val_value_t for the config database to use
*
* RETURNS:
*   status
********************************************************************/
static status_t
    callback_function (ses_cb_t *scb,
                       rpc_msg_t *msg,
                       val_value_t *candidate,
                       val_value_t *running)
{
    log_debug("\n\n Enter validate_compl_callback callback");

    // validate candidate
    // validate running
```

```
    return res;

}  /* callback_function */
```

To unregister callback:

```
   /* Unregister Validate Complete callback */
    agt_cb_validate_complete_unregister(callback_function);
```

7.7.31 SIL-SA Validate Complete Callback

The SIL-SA version of the **Validate Complete** callback function is the user/system callback that is invoked after the **Validate Phase** has been processed during the **<commit>** operation.

The server sends “server request” message to SIL-SA subsystems that registered callbacks right after the **Validate Phase** is processed. Once the SIL-SA subsystems receive the “server-message” they will invoke the callbacks. After the callbacks are invoked and if they are successful then subsystem responds with the “ok” message. If the callbacks are not invoked due to some reason or if the callbacks return error then an “error” message is returned to the server causing the server transaction termination.

The callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback that is intended to provide validation point after the **Validate Phase** is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.

NOTE:

The **<commit>** operation is not available if the default target is set to **<running>**, if the server is run with active **:writable:running** capability.

The callback is only called after commit operation for the specific phase has finished not after the validate for the specific module or edit is done.

The following function template definition is used for **SIL-SA Validate Complete** callback functions:

```
/* Typedef of the SIL-SA validate_complete callback */
typedef status_t
    (*agt_cb_sa_validate_complete_t) (const xmlChar *transaction_id);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_sa_validate_complete**
  - Inputs:
    - **transaction_id** == transaction ID of the transaction control block in progress
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_sa_validate_complete_register**
- Unregister: **agt_cb_sa_validate_complete_unregister**

7.7.32 SIL-SA Validate Complete Callback Examples

The **SIL-SA** version of **Validate Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

All the registered callbacks will be called one after another right after the **Validate Phase** during the **<commit>** operation is done.

The callback is setup in platform-specific way with an API function. To register callback:

```
   /* Register SIL-SA Validate Complete callback */
    res =
        agt_cb_sa_validate_complete_register(hooks_validate_complete_cb);
```

Now, whenever the **Validate Phase** is done for the **<commit>** operation, the callback function will be called.

The callback function could look as follows:

```
/*********************************************************************
* FUNCTION hooks_validate_complete_cb
*
*  SIL-SA Validate Complete callback.
*
*
* RETURNS:
*   status
*********************************************************************/
static status_t
    hooks_validate_complete_cb (const xmlChar *transaction_id_val)
{
    if (!transaction_id_val) {
        log_error("\ntransaction_id value not set");
        return ERR_INTERNAL_VAL;
    }

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();
    status_t res = NO_ERR;

    if (LOGDEBUG2) {
        log_debug2("\n\n*******************************************");
        log_debug2("\nEnter hooks_validate_complete_cb callback");
        log_debug2("\ntransaction_id -- %s", transaction_id_val);
        log_debug2("\nuser_id -- %s", user);
        log_debug2("\nclient_addr -- %s", client_addr);
        log_debug2("\n*******************************************\n\n");
    }

    return res;

}  /* hooks_validate_complete_cb */
```

To unregister callback:

```
/* Unregister SIL-SA Validate Complete callback */
 agt_cb_sa_validate_complete_unregister(hooks_validate_complete_cb);
```

7.7.33 Apply Complete Callback

The **Apply Complete** function is the user/system callback that is invoked after the **Apply Phase** has been processed during the **<commit>** operation.

If **Apply Complete** callback fails the status of the failing callback is returned immediately and no further callbacks are made. As a result, the server will abort the commit.

The **Apply Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback is intended to allow manipulations with the running and candidate configurations (equivalent to completion of apply phase during **<commit>**).

The callback is only called after commit operation for the specific phase has finished not after the apply for the specific module or edit is done.

The following function template definition is used for **Apply Complete** callback functions:

```
/* Typedef of the apply_complete callback */
typedef status_t
      (*agt_cb_apply_complete_t)(ses_cb_t *scb,
                                 rpc_msg_t *msg,
                                 val_value_t *candidate,
                                 val_value_t *running);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_apply_complete**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming rpc_msg_t in progress
    - **candidate** == candidate val_value_t for the config database to use
    - **running** == running val_value_t for the config database to use
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_apply_complete_register**
- Unregister**: agt_cb_apply_complete_unregister**

7.7.34 Apply Complete Callback Examples

The **Apply Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

All the registered callbacks will be called one after another right after the **Apply Phase** during the **<commit>** operation is done.

The callback is setup in platform-specific way with an API function. To register callback:

```
  /* Register Apply Complete callback */
   res = agt_cb_apply_complete_register(callback_function);
```

Now, whenever the apply phase is done for the **<commit>** operation, the callback function will be called and provide candidate and running datastores.

The callback function could look like:

```
/********************************************************************
* FUNCTION  callback_function
*
* Apply Complete callback
* The Apply Complete function is the
* user/system callback that is invoked after
* apply phase has been processed during <commit> op.
*
* Max Callbacks: No limit (except available heap memory)
*
* INPUTS:
*   scb == session control block making the request
*   msg == incoming rpc_msg_t in progress
*   candidate == candidate val_value_t for the config database to use
*   running == running val_value_t for the config database to use
*
* RETURNS:
*   status
********************************************************************/
static status_t
    callback_function (ses_cb_t *scb,
                       rpc_msg_t *msg,
                              val_value_t *candidate,
                              val_value_t *running)
{
    log_debug("\n\n Enter apply_compl_callback callback");

    // validate candidate
    // validate running


    return res;

}  /* callback_function */
```

To unregister callback:

```
/* Unregister Apply Complete callback */
 agt_cb_apply_complete_unregister(callback_function);
```

7.7.35 SIL-SA Apply Complete Callback

The **SIL-SA** version of the **Apply Complete** callback function is the user/system callback that is invoked after the **Apply Phase** has been processed during the **<commit>** operation.

The server sends “server request” message to SIL-SA subsystems that registered callbacks right after the **Apply Phase** is processed. Once the SIL-SA subsystems receive the “server-message” they will invoke the callbacks. After the callbacks are invoked and if they are successful then subsystem responds with the “ok” message. If the callbacks are not invoked due to some reason or if the callbacks return error then an “error” message is returned to the server causing the server transaction termination.

The callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback that is intended to provide validation point after the **Apply Phase** is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.

NOTE:

The **<commit>** operation is not available if the default target is set to **<running>**, if the server is run with active **:writable:running** capability.

The callback is only called after commit operation for the specific phase has finished not after the apply for the specific module or edit is done.

The following function template definition is used for **SIL-SA Apply Complete** callback functions:

```
/* Typedef of the SIL-SA apply_complete callback */
typedef status_t
    (*agt_cb_sa_apply_complete_t) (const xmlChar *transaction_id);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_sa_apply_complete**
  - Inputs:
    - **transaction_id** == transaction ID of the transaction control block in progress
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_sa_apply_complete_register**
- Unregister: **agt_cb_sa_apply_complete_unregister**

7.7.36 SIL-SA Apply Complete Callback Examples

The **SIL-SA** version of **Apply Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

All the registered callbacks will be called one after another right after the **Apply Phase** during the **<commit>** operation is done.

The callback is setup in platform-specific way with an API function. To register callback:

```
   /* Register SIL-SA Apply Complete callback */
    res =
        agt_cb_sa_apply_complete_register(hooks_apply_complete_cb);
```

Now, whenever the **Apply Phase** is done for the **<commit>** operation, the callback function will be called.

The callback function could look as follows:

```
/*********************************************************************
* FUNCTION hooks_apply_complete_cb
*
*  SIL-SA Apply Complete callback.
*
*
* RETURNS:
*   status
*********************************************************************/
static status_t
    hooks_apply_complete_cb (const xmlChar *transaction_id_val)
{
    if (!transaction_id_val) {
        log_error("\ntransaction_id value not set");
        return ERR_INTERNAL_VAL;
    }

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();
    status_t res = NO_ERR;

    if (LOGDEBUG2) {
        log_debug2("\n\n*****************************************");
        log_debug2("\nEnter hooks_apply_complete_cb callback");
        log_debug2("\ntransaction_id -- %s", transaction_id_val);
        log_debug2("\nuser_id -- %s", user);
        log_debug2("\nclient_addr -- %s", client_addr);
        log_debug2("\n*****************************************\n\n");
    }

    return res;

}  /* hooks_apply_complete_cb */
```

To unregister callback:

```
/* Unregister SIL-SA Apply Complete callback */
 agt_cb_sa_apply_complete_unregister(hooks_apply_complete_cb);
```

7.7.37 Commit Complete Callback

The **Commit Complete** callback is a user/system callback that is invoked when the **<commit>** operation completes without errors or the internal **<replay-config>** operation completes without errors.

If a **Commit Complete** callback fails the status of the failing operation is returned immediately and no further **Commit Complete** callbacks are called.

The callback is only called after commit operation for the specific phase has finished not after the commit for the specific module or edit is done.

Note: There is not such an operation as **<commit>** if the default target is set tot **<running>**. Thus, the **Commit Complete** callback is not going to be available if the server is run with active :writable:running capability.

Callback Template

- Type: User Callback
- Max Callbacks: none
- File: **agt/agt_commit_complete.h**
- Template: **agt_commit_complete_cb_t**
  - Inputs:
    - **commit_type** == The type of commit that was just completed:

      AGT_COMMIT_TYPE_NORMAL: <commit> operation was completed

      AGT_COMMIT_TYPE_REPLAY: <replay-commit> operation was completed
- Outputs: none
- Returns: **status_t:**

  Status of the callback function execution
- Register: **agt_commit_complete_register**
- Unregister: **agt_commit_complete_unregister**

The following function template definition is used for **Commit Complete** callback functions:

```
/* Typedef of the callback */
typedef status_t
    (*agt_commit_complete_cb_t)(agt_commit_type_t commit_type);
```

7.7.38 Commit Complete Callback Examples

The **Commit Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

The callback is setup in platform-specific way with an API function. This function registers a commit-complete callback that will be called after all changes to the candidate database have been committed. The function will be called after that final SIL commit operation. If a commit complete operation is already registered for the module it will be replaced.

Only one active callback per SIL code can be assigned. That's if you register one callback and then trying to register another for the same module, the first one will be deactivated and the second will become active.

All the registered callbacks will be called one after another right after the **Commit Phase** during the commit operation is done.

To register callback:

```
#define EXAMPLE_MOD (const xmlChar *)"example"

   /* Register Commit Complete callback */
    res = agt_commit_complete_register(EXAMPLE_MOD, callback_function);
```

Now, whenever the commit phase is done for the **<commit>** operation, the callback function will be called.

The callback function could look like:

```
/********************************************************************
* FUNCTION  callback_function
*
* Commit Complete callback
* The Commit Complete function is the
* user/system callback that is invoked after
* the commit phase has been processed during <commit> op.
*
* INPUTS:
*   commit_type == enum identifying commit type (normal or replay)
*
* RETURNS:
*   status
********************************************************************/
static status_t
    commit_compl_callback (agt_commit_type_t commit_type)
{
    log_debug("\n\n Enter commit_compl_callback callback");


    return res;

}  /* callback_function */
```

To unregister callback:

```
/* Unregister Commit Complete callback */
 agt_commit_complete_unregister(EXAMPLE_MOD);
```

7.7.39 SIL-SA Commit Complete Callback

The SIL-SA version of the **Commit Complete** callback function is the user/system callback that is invoked after the **Commit Phase** has been processed during the **<commit>** operation.

The server sends “server request” message to SIL-SA subsystems that registered callbacks right after the **Commit Phase** is processed. Once the SIL-SA subsystems receive the “server-message” they will invoke the callbacks. After the callbacks are invoked and if they are successful then subsystem responds with the “ok” message. If the callbacks are not invoked due to some reason or if the callbacks return error then an “error” message is returned to the server causing the server transaction termination.

The callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback that is intended to provide validation point after the **Commit Phase** is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.

NOTE:

The **<commit>** operation is not available if the default target is set to **<running>**, if the server is run with active **:writable:running** capability.

The callback is only called after commit operation for the specific phase has finished not after the commit for the specific module or edit is done.

The following function template definition is used for **SIL-SA Commit Complete** callback functions:

```
/* Typedef of the SIL-SA commit_complete callback */
typedef status_t
    (*agt_cb_sa_commit_complete_t) (const xmlChar *transaction_id,
                                    agt_commit_type_t commit_type);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_sa_commit_complete**
  - Inputs:
    - **transaction_id** == transaction ID of the transaction control block in progress
    - **commit_type** == commit type:
      - **AGT_COMMIT_TYPE_REPLAY**: <replay-commit> operation was completed
      - **AGT_COMMIT_TYPE_NORMAL**: <commit> operation was completed
- Outputs: none
- Returns: Status of the callback function execution

- Register: **agt_cb_sa_commit_complete_register**
- Unregister: **agt_cb_sa_commit_complete_unregister**

7.7.40 SIL-SA Commit Complete Callback Examples

The **SIL-SA** version of **Commit Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

All the registered callbacks will be called one after another right after the **Commit Phase** during the **<commit>** operation is done.

The callback is setup in platform-specific way with an API function. To register callback:

```
  /* Register SIL-SA Commit Complete callback */
   res =
       agt_cb_sa_commit_complete_register(hooks_commit_complete_cb);
```

Now, whenever the **Commit Phase** is done for the **<commit>** operation, the callback function will be called.

The callback function could look as follows:

```
/********************************************************************
* FUNCTION hooks_commit_complete_cb
*
*  SIL-SA Commit Complete callback.
*
*
* RETURNS:
*   status
********************************************************************/
static status_t
    hooks_commit_complete_cb (const xmlChar *transaction_id_val,
                              agt_commit_type_t commit_type)
{
    if (!transaction_id_val) {
        log_error("\ntransaction_id value not set");
        return ERR_INTERNAL_VAL;
    }

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();
    const xmlChar *type = agt_commit_complete_get_type(commit_type);
    status_t res = NO_ERR;
    status_t res2 = NO_ERR;

    if (commit_type == AGT_COMMIT_TYPE_REPLAY) {
        return res;
    }

    if (LOGDEBUG2) {
        log_debug2("\n\n*******************************************");
```

```
        log_debug2("\nEnter hooks_commit_complete_cb callback");
        log_debug2("\ntransaction_id -- %s", transaction_id_val);
        log_debug2("\nuser_id -- %s", user);
        log_debug2("\nclient_addr -- %s", client_addr);
        log_debug2("\ncommit type -- %s", type);
    }

    const xmlChar *defpath =
        (const xmlChar *)"/testsystem";

    val_value_t *testval =
        sil_sa_get_data(NCX_CFGID_CANDIDATE,
                        defpath,
                        &res2);
    if (testval) {
        log_info("\n+++ Got testval %s (%s)",
                 VAL_NAME(testval),
                 get_error_string(res));

        val_value_t *child = val_get_first_child(testval);
        if (child) {
            if (child->btyp && typ_is_simple(child->btyp)) {
                xmlChar *valstr = val_make_sprintf_string(child);
                if (valstr == NULL) {
                    return ERR_INTERNAL_MEM;
                }

                log_debug("\nchild:   %s='%s'",
                          VAL_NAME(child),
                          valstr);

                /* FORCE ERROR */
                if (!xml_strcmp(valstr,
                        (const xmlChar *)"commit-completeness-test1")) {

                    res = ERR_NCX_OPERATION_NOT_SUPPORTED;
                    log_info("\n------ REPORTING AN ERROR (%s)\n",
                        get_error_string(res));
                }

                m__free(valstr);
            }
        }
    }

    if (LOGDEBUG2) {
        log_debug2("\n*******************************************\n\n");
    }

    return res;

}  /* hooks_commit_complete_cb */
```

To unregister callback:

```
  /* Unregister SIL-SA Commit Complete callback */
   agt_cb_sa_commit_complete_unregister(hooks_commit_complete_cb);
```

7.7.41 Rollback Complete Callback

The **Rollback Complete** function is the user/system callback that is invoked after and if the **Rollback Phase** has been processed during the **<commit>** operation.

If **Rollback Complete** callback fails the status of the failing callback is returned immediately and no further callbacks are made. As a result, the server will report an error.

The **Rollback Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback is intended to allow manipulations with the running and candidate configurations after the **Rollback Phase.**

The following function template definition is used for **Rollback Complete** callback functions:

```
/* Typedef of the rollback_complete callback */
typedef status_t
      (*agt_cb_rollback_complete_t)(ses_cb_t *scb,
                                    rpc_msg_t *msg,
                                    val_value_t *candidate,
                                    val_value_t *running);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_rollback_complete**
  - Inputs:
    - **scb** == session control block making the request
    - **msg** == incoming rpc_msg_t in progress
    - **candidate** == candidate val_value_t for the config database to use
    - **running** == running val_value_t for the config database to use
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_rollback_complete_register**
- Unregister: **agt_cb_rollback_complete_unregister**

7.7.42 SIL-SA Rollback Complete Callback

The **SIL-SA** version of the **Rollback Complete** callback function is the user/system callback that is invoked after the **Rollback Phase** has been processed during the **<commit>** operation.

The server sends “server request” message to SIL-SA subsystems that registered callbacks right after the **Rollback Phase** is processed. Once the SIL-SA subsystems receive the “server-message” they will invoke the callbacks. After the callbacks are invoked and if they are successful then subsystem responds with the “ok” message. If the callbacks are not invoked due to some reason or if the callbacks return error then an “error” message is returned to the server causing the server transaction termination.

The callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

The callback that is intended to provide validation point after the **Rollback Phase** is completed. For SIL-SA usage the server does not provide running and candidate values, instead it passes the transaction ID for reference and **Get Data** API can be used to run additional validation if required.

NOTE:

The **<commit>** operation is not available if the default target is set to **<running>**, if the server is run with active **:writable:running** capability.

The callback is only called after commit operation for the specific phase has finished not after the rollback for the specific module or edit is done.

The following function template definition is used for **SIL-SA Rollback Complete** callback functions:

```
/* Typedef of the SIL-SA rollback_complete callback */
typedef status_t
    (*agt_cb_sa_rollback_complete_t) (const xmlChar *transaction_id);
```

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_sa_rollback_complete**
  - Inputs:
    - **transaction_id** == transaction ID of the transaction control block in progress
- Outputs: none
- Returns: Status of the callback function execution
- Register: **agt_cb_sa_rollback_complete_register**
- Unregister: **agt_cb_sa_rollback_complete_unregister**

7.7.43 SIL-SA Rollback Complete Callback Examples

The **SIL-SA** version of **Rollback Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it is done for EDIT or GET callbacks.

Example:

All the registered callbacks will be called one after another right after the **Rollback Phase** during the **<commit>** operation is done.

The callback is setup in platform-specific way with an API function. To register callback:

```
   /* Register SIL-SA Rollback Complete callback */
    res =
        agt_cb_sa_rollback_complete_register(hooks_rollback_complete_cb);
```

Now, whenever the **Rollback Phase** is done for the **<commit>** operation, the callback function will be called.

The callback function could look as follows:

```
/********************************************************************
* FUNCTION hooks_rollback_complete_cb
*
*  SIL-SA Rollback Complete callback.
*
*
* RETURNS:
*   status
*********************************************************************/
static status_t
    hooks_rollback_complete_cb (const xmlChar *transaction_id_val)
{
    if (!transaction_id_val) {
        log_error("\ntransaction_id value not set");
        return ERR_INTERNAL_VAL;
    }

    const xmlChar *user = sil_sa_get_username();
    const xmlChar *client_addr = sil_sa_get_client_addr();
    status_t res = NO_ERR;

    if (LOGDEBUG2) {
        log_debug2("\n\n*****************************************");
        log_debug2("\nEnter hooks_rollback_complete_cb callback");
        log_debug2("\ntransaction_id -- %s", transaction_id_val);
        log_debug2("\nuser_id -- %s", user);
        log_debug2("\nclient_addr -- %s", client_addr);
        log_debug2("\n*****************************************\n\n");
    }

    return res;

}  /* hooks_rollback_complete_cb */
```

To unregister callback:

```
/* Unregister SIL-SA Rollback Complete callback */
 agt_cb_sa_rollback_complete_unregister(hooks_rollback_complete_cb);
```

7.7.44 Dynamic Default Hook Callback

The **Dynamic Default Hook** function is the user/system callback that can be used to set up dynamic default value to the non-default leafy nodes during load or edit-config operations. It is invoked when the server checks if the node has any defaults and if there is no any YANG defined defaults the server can update the node with custom “system” default value.

NOTE:

The **Dynamic Default Hook** callback is NOT to overwrite a YANG default value, it is only for the nodes that do not have a default statement defined. The YANG default values can only be changed statically with help of deviations. The **Dynamic Default Hook** callback should be used to dynamically change a value of the node. This callback will NOT make the value of the leaf or leaf-list to be default value as if YANG default statement sets it.

If the callback fails and return an error the transaction will be terminated and the server will generate an error. However, the **ERR_NCX_SKIPPED** or **ERR_NCX_NO_INSTANCE** status will cause the serve to skip over the callback node and do not terminate the transaction.

The **Dynamic Default Hook** callbacks will be invoked during the following operations:

- **Load startup configuration:** the server will try to add defaults for all the nodes in the startup configuration and in addition will try to invoke **Dynamic Default Hook** callbacks to set up custom dynamic default nodes if any;
- **EDIT configurations:** the server will try to add defaults for all the nodes in the <edit-config> and in addition will try to invoke **Dynamic Default Hook** callbacks to set up custom dynamic default nodes if any.

The **Dynamic Default Hook** callbacks specifications and limitations:

- Allowed only for leaf or leaf-list nodes;
- Not allowed for keys nodes;
- Not allowed for RPC/ACTION input nodes;
- Not invoked on deletion of any nodes;
- No support for **yangdump-pro** auto-generated code;
- No support for SIL-SA;
- **Only one dynamic default value is allowed for leaf-list nodes.**

The following function template definition is used for **Dynamic Default Hook** callback functions:

```
/********************************************************************
* Typedef of the ncx_def_hook_cbfn_t callback
*
* The Dynamic Default Hook function is the user/system callback
* that is invoked before the server checks if the node has
* any defaults and if there is not any YANG defined defaults
* the server can update the node with custom "system" default
* value.
*
* Run an instrumentation-defined function
* for val set default event
*
* INPUTS:
*   parentval == parent val_value node to use
*   obj == object template to use
*
* OUTPUTS
*   *buff == malloced buffer that represents a new default value
*
* RETURNS:
*   status: ERR_NCX_SKIPPED or ERR_NCX_NO_INSTANCE will cause the server
*           to skip over this node. Otherwise the server will report an
*           error and ternimate the transaction
*
*********************************************************************/
typedef status_t
    (*ncx_def_hook_cbfn_t) (struct val_value_t_ *parentval,
                            struct obj_template_t_ *obj,
                            xmlChar **buff);
```

Callback Template

- Type: User Callback
- Max Callbacks: 1 per object
- File: **ncxtypes.h**
- Template: **ncx_def_hook_cbfn_t**
  - Inputs:
    - **parentval** == parent val_value node to use
    - **obj** == object template to use
  - Outputs:
    - ***buff** == malloced buffer that represents a new default value
- Returns: status:
  - ERR_NCX_SKIPPED or ERR_NCX_NO_INSTANCE will cause the server to skip over this node.
  - Otherwise the server will report an error and terminate the transaction
- Register: **agt_cb_def_hook_register**
- Unregister: **agt_cb_def_hook_unregister**

7.8 Notifications

The **yangdump-sdk** program will automatically generate functions to queue a specific notification type for processing. It is up to the SIL callback code to invoke this function when the notification event needs to be generated. The SIL code is expected to provide the value nodes that are needed for any notification payload objects. SIL-SA uses a different function to queue a notification than SIL code, but all other APIs are the same.

SIL or SIL-SA code can include an Event Stream Callback function to know if any clients are listening on the relevant event stream. If no subscriptions are active, then the notification send processing logic can decide to suppress generation of certain events.

7.8.1 Notification Send Function

The function to generate a notification control block and queue it for notification replay and delivery is generated by the **yangdump-pro** program. A function parameter will exist for each top-level data node defined in the YANG notification definition.

In the example below, the 'toastDone' notification event contains just one leaf, called the 'toastStatus'. There is SIL timer callback code which calls this function, and provides the final toast status, after the <make-toast> operation has been completed or canceled.

```
/********************************************************************
* FUNCTION y_toaster_toastDone_send
*
* Send a y_toaster_toastDone notification
* Called by your code when notification event occurs
*
*********************************************************************/
void
    y_toaster_toastDone_send (
        const xmlChar *toastStatus)
{
    agt_not_msg_t *notif;
    val_value_t *parmval;
    status_t res;

    res = NO_ERR;

    if (LOGDEBUG) {
        log_debug("\nGenerating <toastDone> notification");
    }

    notif = agt_not_new_notification(toastDone_obj);
    if (notif == NULL) {
        log_error("\nError: malloc failed, cannot send <toastDone> notification");
        return;
    }

    /* add toastStatus to payload */
    parmval = agt_make_leaf(
        toastDone_obj,
        y_toaster_N_toastStatus,
        toastStatus,
        &res);
    if (parmval == NULL) {
```

```
        log_error(
            "\nError: make leaf failed (%s), cannot send <toastDone> notification",
            get_error_string(res));
    } else {
        agt_not_add_to_payload(notif, parmval);
    }

    agt_not_queue_notification(notif);

    /* !!!  Note that for SIL-SA code the following line is
     * !!!  generated instead of agt_not_queue_notification
     * !! ! sil_sa_queue_notification(notif);
     */

} /* y_toaster_toastDone_send */
```

7.8.2 Notification Send Function Variants

There are 4 API functions for sending notifications:

- Send notification for SIL
- Send notification for SIL-SA
- Send notification to custom event stream for SIL
- Send notification to custom event stream for SIL-SA

1) Send to the default or configured event stream for SIL: agt_not_queue_notification

```
/********************************************************************
* FUNCTION agt_not_queue_notification
*
* Queue the specified notification in the replay log.
* It will be sent to all the active subscriptions
* as needed.
*
* INPUTS:
*   notif == notification to send
*            !!! THIS IS LIVE MALLOCED MEMORY PASSED OFF
*            !!! TO THIS FUNCTION.  IT WILL BE FREED LATER
*            !!! DO NOT CALL agt_not_free_notification
*            !!! AFTER THIS CALL
*
* OUTPUTS:
*   message added to the notificationQ
*
*********************************************************************/
extern void
    agt_not_queue_notification (agt_not_msg_t *notif);
```

2) Send to the default or configured event stream for SIL-SA: sil_sa_queue_notification

```
/********************************************************************
```

```
* FUNCTION sil_sa_queue_notification
*
* Send a notification-event to the main server for queing
* by the agt_not module
* INPUTS:
*   notif == notification struct to send
*   will be consumed and freed even if there is an error
*********************************************************************/
extern void
    sil_sa_queue_notification (agt_not_msg_t *notif);
```

3) Send to ta specified event stream for SIL: agt_not_queue_notification_stream

```
/*********************************************************************
* FUNCTION agt_not_queue_notification_stream
*
* Queue the specified notification in the replay log.
* It will be sent to all the active subscriptions
* as needed.
*
* INPUTS:
*   stream_name == stream name to use (NULL == use NETCONF)
*   notif == notification to send
*            !!! THIS IS LIVE MALLOCED MEMORY PASSED OFF
*            !!! TO THIS FUNCTION.  IT WILL BE FREED LATER
*            !!! DO NOT CALL agt_not_free_notification
*            !!! AFTER THIS CALL; FREED EVEN IF ERROR
*
* OUTPUTS:
*   message added to the notificationQ
*********************************************************************/
extern void
    agt_not_queue_notification_stream (const xmlChar *stream_name,
                                       agt_not_msg_t *notif);
```

4) Send to ta specified event stream for SIL-SA: sil_sa_queue_notification_stream

```
/*********************************************************************
* FUNCTION sil_sa_queue_notification_stream
*
* Send a notification-event to the main server for queing
* by the agt_not module
* INPUTS:
*   stream_name == optional stream name (NULL = use NETCONF)
*   notif == notification struct to send
*   will be consumed and freed even if there is an error
*********************************************************************/
extern void
    sil_sa_queue_notification_stream (const xmlChar *stream_name,
                                      agt_not_msg_t *notif);
```

7.8.3 Event Stream Callback Functions

This callback type is documented in the YumaPro API QuickStart Guide.

7.9 Operational Data

Operational data is data that is set as “config false” in the YANG file.

Operational state and statistics are common forms of operational data.

These nodes are supported within SIL code in three ways:

1. object template based “GET2” callback
2. value node based static operational data
3. value node based virtual operational data

7.9.1 Get2 Object Template Based Operational Data

This API uses 1 callback per object template. The server will issue “get” and “getnext” requests via the API, and the callback code will return value nodes for the data is is managing. These data nodes are not maintained in the server data tree. They are not cached in the server.

Local SIL and remote SIL-SA callbacks are supported for operational data. Configuration data is not supported at this time. These callbacks are expected to return some or all of the terminal child nodes (leaf, leaf-list, anyxml) when invoked.

These callbacks never return child nodes that are complex nodes. Refer to All In One (AIO) GET2 callback for complex node return.

Complex child nodes are expected to have their own GET2 callbacks registered except AIO GET2 callbacks. The GET2 callbacks are registered with a new API similar to the edit callback registration.

The following node types are expected to have GET2 callbacks registered

- choice: expected to return the name of the active case and terminal nodes from the active case
- container: expected to return child terminal nodes
- list: expected to return list keys and maybe other terminal nodes

Callbacks for terminal nodes are allowed, but only if their parent is a **config=true** node. The server will expect configuration data nodes to be present in the target datastore. If the retrieval request includes **config=false** nodes, then the server will check each child node that is **config=false** for a GET2 callback function. If it exists, then it will be used for retrieval of that data node. If not, the current **config=true** node will be checked for child nodes (static or dynamic operational data).

Each GET2 callback returns a status code

- NO_ERR: data is returned
- ERR_NCX_NO_INSTANCE: the specified data instance does not exist
- Other status code: some error occurred

YANG Node Type	GET2 Callback Summary
container	Returns the child terminal nodes in the container
list	Returns the keys and perhaps the rest of the child terminal nodes. Set “more_data” to trigger “getnext” until no more instances found
anyxml	Returns the entire anyxml value tree
leaf	Returns the leaf value
leaf-list	Returns all values for the leaf-list
choice	Returns the name of the active case and the child terminal nodes in the active case
case	This node type does not have a callback

Procedure

The server will send a **<get-request>** server request message to each subsystem that has registered a GET2 callback for the specific object.

- For single-instance nodes (container, leaf, choice, anyxml) only a “get” request will be made.
- For a leaf-list node, only “get” requests will be made, but all instances of the leaf-list are returned at once (max-entries parameter equals zero).
- For a list node, a “get” request will be made if the keys are known. A “get” request that contains no keys will cause the first instance to be returned. Entries are ordered by their key values, exactly as specified in the YANG list. If there is no key-stmt defined, then the callback will still return the first entry, except the server will not attempt any sorting if multiple entries are returned (E.g 1 from each subsystem). It is always up to the callback to know what is the “best” next entry, given the keys that are returned.
- The callback examined the keys (if any) and fills in the return_keys. If the “keys-only” flag is set, then no other terminal nodes are returned.
- If there are more list entries, then the “more-data” flag is set in the get-response.
- The server will issue “getnext” requests as needed until the “more-data” flag is not present in the response.
- The “max-entries” field will be set to 1. It may be set higher in a future release to support getbulk retrieval.

GET2 Callback API

The file **netconf/src/ncx/getcb.h** defines the GET2 callback function:

```
/* Callback function for server object handler get2 callback
 * Used to provide main and/or subsystem retrieval of instances
 * of a specific named object
 *
 * INPUTS:
 *   scb == session control block making the request
 *   msg == incoming XML message header
 *   get2cb == get2 control block for this callback request
 *
```

```
 * OUTPUTS:
 *   return_keyQ is full of any new keys added for this
 *   entry -- only if 'obj' is a list
 *   return_valQ is filled with malloced val_value_t nodes
 *   If object tested is a choice
 *   the a backptr to a constant string containing the case
 *   name that is active
 *
 * RETURNS:
 *    status:
 *      NO_ERR if executed OK and found OK
 *      ERR_NCX_NO_INSTANCE warning if no instance found
 */
typedef status_t
    (*getcb_fn2_t) (ses_cb_t *scb,
                    xml_msg_hdr_t *msg,
                    getcb_get2_t *get2cb);
```

getcb_get2_t Structure

This data structure contains the fields used by the server and the callback to exchange data.

- **getcb Request Fields**: set by **getcb** before the GET2 callback is invoked
  - **txid_str:** GET2 transaction IF string
    - GETCB_GET2_TXID_STR(cb) access macro
  - **obj:** pointer to obj_template_t for the object being retrieved
    - GETCB_GET2_OBJ(cb) access macro
  - **testfn:** test filter function for checking nodes before processing (may be NULL)
    - GETCB_GET2_TESTFN access macro
  - **keyQ**: queue of val_value_t structs representing the key leafs for the request. These are keys for the ancestor-or-self lists for this object.
    - Fixed keys are identified using the VAL_IS_FIXED_VALUE(keyval) macro
    - The GET2 callback must only return entries matching the provided key values
    - If no value is provided for a key then the first value is requested
    - GETCB_GET2_KEYQ(cb) access macro
    - GETCB_GET2_FIRST_KEY(cb) macro to return the first val_value_t in the queue
    - GETCB_GET2_NEXT_KEY(cb, cur) macro to return the next val_value_t in the queue
  - **matchQ**: queue of val_value_t structs representing non-key leaf content-match nodes
    - These are always simple child nodes of the object identified by 'obj'
    - The content-match test is optional to implement by the GET2 callback
    - If the content-match tests have been performed by the GET2 callback, then the return flag is set with the GETCB_GET2_CONTENT_MATCH_DONE() macro
    - All entries in the matchQ must match the entry being returned. If not, then an ERR_NCX_NO_INSTANCE status is returned instead
    - GETCB_GET2_MATCHQ(cb) access macro
    - GETCB_GET2_FIRST_MATCH(cb) macro to return the first val_value_t in the queue

        - GETCB_GET2_NEXT_MATCH(cb, cur) macro to return the next val_value_t in the queue
    - **cbmode**: the GET2 callback mode
        - GETCB_GET_VALUE for get-exact
        - GETCB_GETNEXT_VALUE for get-next
        - GETCB_GET2_CBMODE(cb) access macro
    - **max_entries**: maximum number of values to return (for getnext mode)
        - 0 == return all entries
        - 1 – N == return at most this number of entries
            - Currently set to 1 by the server
        - GETCB_GET2_MAX_ENTRIES(cb) access macro
    - **max_levels**: maximum depth of subtrees parameter to determine if the maximum level has been reached
        - GETCB_GET2_MAX_LEVELS(cb) access macro
    - **oper_data:** true if config=false data should be returned
        - GETCB_GET2_OPER_DATA(cb) access macro
    - **config_data:** true if config=true data should be returned
        - GETCB_GET2_CONFIG_DATA(cb) access macro
    - **expand_varexpr:** true is variable expressions should be expanded; false to print the variable instead
        - GETCB_GET2_EXPAND_VAREXPR(cb) access macro
    - **keys_only**: true if only list keys should be returned; false for all terminal nodes
        - GETCB_GET2_KEYS_ONLY(cb) access macro
    - **with_defaults**: true if default nodes should be returned; false if they are not requested
        - GETCB_GET2_WITH_DEFAULTS(cb) access macro
    - **api_mode:** the consumer walker callback API mode
        - GETCB_API_MODE_NORMAL: entries walked with BEGIN, TERM, END
        - GETCB_API_MODE_1SHOT: entry called with 1SHOT, no child nodes walked
        - GETCB_GET2_API_MODE(cb) access macro
- **getcb Response Fields:** set by the GET2 callback before it returns
    - **more_data:** set to true if there are more instances of this object to retrieve; false otherwise
    - GETCB_GET2_MORE_DATA(cb) access macro
    - **match_test_done**: set to true if the GET2 callback performed the request content match test(s); false if not or no match tests requested
        - GETCB_GET2_MATCH_TEST_DONE(cb) access macro
    - **active_case_modname**: name of the modue for the module namespace of the active case; set to NULL for the same module name as the choice
        - Ignored unless the 'obj' type is 'choice'
        - GETCB_GET2_ACTIVE_CASE_MODNAME(cb) access macro
    - **active_case:** name of the active case

        - Ignored unless the 'obj' type is 'choice'
        - GETCB_GET2_ACTIVE_CASE(cb) access macro
    - **return_keyQ:** queue of val_value_t structs representing the key leafs of the returned value
        - Values from the keyQ can be transferred to this queue
        - GETCB_GET2_RETURN_KEYQ(cb) access macro
        - GETCB_GET2_FIRST_RETURN_KEY(cb) macro to return the first val_value_t in the queue
        - GETCB_GET2_NEXT_RETURN_KEY(cb, cur) macro to return the next val_value_t in the queue
    - **return_valQ:** queue of val_value_t structs representing the non-key terminal nodes of the returned value
        - Only the first value is used at this time, except for leaf-list nodes
        - Should be empty if the 'keys-only' flag is set in the request
        - GETCB_GET2_RETURN_VALQ(cb) access macro
        - GETCB_GET2_FIRST_RETURN_VAL(cb) macro to return the first val_value_t in the queue
        - GETCB_GET2_NEXT_RETURN_VAL(cb, cur) macro to return the next val_value_t in the queue
    - **return_val:** inline val_value_t to return a single leaf
        - Can be used instead of the return_valQ to avoid mallocing a val_value_t
        - GETCB_GET2_RETURN_VAL(cb) access macro
        - GETCB_GET2_RETURN_VAL_SET(cb) macro to test if the value is set
    - **return_aioQ:** queue of val_value_t structure representing the All In One (AIO) nodes of the returned value
        - GETCB_GET2_RETURN_AIOQ(cb) access macro
        - GETCB_GET2_FIRST_RETURN_AIO_VAL(cb) macro to return the first val_value_t in the queue
        - GETCB_GET2_NEXT_RETURN_AIO_VAL(cur) macro to return the next val_value_t in the queue
    - **aio_return_buff:** AIO XML or JSON malloced buffer from callback to use instead of val_value_t structures
        - GETCB_AIO_BUFFER(cb) access macro
        - GETCB_AIO_ENCODING(cb) macro to access AIO callback encoding

The GETCB_GET2_ macros in **getcb.h** are used to access this data structure.

There are also several API functions in **getcb** to access this data as well:

```
/* get2 control block */
typedef struct getcb_get2_t_ {
    dlq_hdr_t                  qhdr;

    /**********  I N P U T  P A R A M E T E R S   *******/

    /* transaction ID string */
    xmlChar                    *txid_str;

    /* object template containing this callback */
    obj_template_t             *obj;

    /* value node test function (may be obsolete for get2) */
    val_nodetest_fn_t          testfn;
```

```
    /* Q of malloced val_value_t; 1 entry for each key leaf
     * includes the ancestor keys and keys for the current list
     * (if applicable)
     * If the set of keys for the current list is incomplete,
     * then the provided keys are 'fixed'
     */
    dlq_hdr_t                   keyQ;

    /* Q of malloced val_value_t; 1 entry for each content-match
     * leaf in the subtree or XPath filter; these leafs only
     * apply to the current object
     */
    dlq_hdr_t                   matchQ;

    /* Q of malloced getcb_get2_select_t; 1 entry for each
     * child select node of the object in this get2cb
     * these select nodes only apply to the current object
     * if the 'select_only' flag is set then this queue is used
     *
     * An empty selectQ means only keys should be returned (for a list)
     * For a choice the active case must be set.
     * For a P-container, the callback has to return NO_ERR instead
     * of ERR_NCX_NO_INSTANCE
     *
     * The get2 callback can ignore the select_only flag and selectQ
     * The extra returned values will be ignored by the caller.
     */
    dlq_hdr_t                   selectQ;  /* Q of getcb_get2_select_t */

    /* reason for the callback (get or getnext) */
    getcb_mode_t                cbmode;

    /* max instances to get
     * 0 for all entries, 1 .. N for specific max
     */
    uint32                      max_entries;

    /* 0 for all levels, 1 .. N for max
     * max_levels forced to 1 if testmode=TRUE
     */
    uint32                      max_levels;

    /* TRUE to get operational data */
    boolean                     oper_data;

    /* TBD: TRUE to get configuration data
     * not supported yet! All config must be in
     * the cfg->root val_value_t tree
     */
    boolean                     config_data;

    /* variable expressions: TRUE if a varexpr node should
     * be expanded; FALSE if printed as an expression
     */
    boolean                     expand_varexpr;

    /* keys-only: TRUE if only the key leafs are desired from
     * list objects; FALSE if normal retrieval
     */
    boolean                     keys_only;

    /* select: TRUE if only the selectQ child nodes are desired from
```

```
     * the parent for this callback; FALSE if no select mode retrieval
     */
    boolean                     select_only;

    /* with_defaults: TRUE if default nodes should be returned
     * this is needed for operational data because the server does
     * not process when-stmts on operational data, so the instrumentation
     * has to return operational defaults if they are requested
     */
    boolean                     with_defaults;

    /* api_mode: consumer API callback mode
     *   GETCB_API_MODE_NORMAL:
     *      normal walk through objects including subtrees
     *      complex nodes are walked START, TERM*, END
     *      simple nodes are walked TERM
     *   GETCB_API_MODE_1SHOT:
     *      the specified object is retrieved according to
     *      the keys and content-match nodes and the
     *      consumer callback is called only one
     *      The callbacks will not include nested subtrees
     *   GETCB_API_MODE_CHOICE:
     *      The getcb code is retrieving implied choice-stmt
     *      active-case values and possibly terminal values
     */
    getcb_api_mode_t            api_mode;

    /* get-request with-origin flag */
    boolean                     with_origin;

    /**********  I N T E R N A L   S T A T E   D A T A   *******/

    /* first key val node that was moved from the return_keyQ
     * to the keyQ for nested nodes to process; needs to be removed
     * after each nested list is processed
     */
    val_value_t                 *start_add_key;
    boolean                     acmtest_result;
    boolean                     last_sibling;
    boolean                     isfirst_nokey;
    boolean                     first_llsibling; // leaf-list siblings
    boolean                     last_llsibling;  // leaf-list siblings

    /* Used only for AIO RESTCONF processing */
    boolean                     islast;   // also used for JSON subtree proc
    boolean                     isfirst;  // also used for JSON subtree proc
    boolean                     aio_done;

    /* Used for JSON subtree proccessing */
    boolean                     first_child;
    boolean                     first_sibling;
    boolean                     force_lastsibling;
    boolean                     force_lastsib_value;
    boolean                     force_array_obj;

    /**********  C A L L B A C K   R E T U R N   D A T A   *******/

    /* set by the callback function if there are more instances
     * that follow the specified instance; this is a global flag
     * that applies to the entire response
     */
    boolean                     more_data;
```

```
    /* set by the callback function if this is a list callback
     * and multiple entries are returned; this is a global queue
     * that applies to the entire response
     */
    dlq_hdr_t                   getbulkQ;   /* Q of getcb_get2_getbulk_t */

    /* set by a choice test_mode callback to return the name of
     * the active case; If the active_case_modname is NULL then
     * the module name of the parent choice-stmt will be used
     */
    xmlChar                     *active_case_modname;
    xmlChar                     *active_case;

    /* Q of malloced val_value_t */
    dlq_hdr_t                   return_keyQ;

    /* Q of malloced val_value_t */
    dlq_hdr_t                   return_valQ;

    /* if just 1 instance is returned, then the return_val
     * will be used; if the btyp is set to something other
     * than NCX_BT_NONE, then the server will use this
     * value; otherwise 1 or more malloced val_value_t structs
     * are expected in the return_valQ
     *
     * THIS STRUCT MUST NOT BE USED IN GETBULK MODE
     */
    val_value_t                 return_val;

    /* if this is a request that causes multiple responses
     * then the responseQ will have each response get2cb
     */
    dlq_hdr_t                   responseQ;   // Q of getcb_get2_t structs

    /* content-match done flag
     * ignored unless the matchQ is non-empty
     * If TRUE. indicates that the callback performed the requested
     * content-match tests and the returned output is post-filter
     * If FALSE, then the content-match tests were not done
     */
    boolean                     match_test_done;

    /* If TRUE then Last list failed BUT some of the entries were
     * written successfully. So need to write comma after this
     * list object.
     */
    boolean                     wrote_some_lists;

    /* save parent backtrs for when-stmt processing of GET2 data */
    val_value_t                 *parent_val;

    struct getcb_get2_t_        *parent_cb;

    /* Q of malloced val_value_t.
     * Used for sil-aio-get2 extension only.
     * Callbacks will fill in data into this Queue.
     * The callback will be called only once and the server will
     * expect the return_aioQ to be filled in with the whole
     * set of children.
     */
    dlq_hdr_t                   return_aioQ;

    /* save NMDA datastore for GET operational */
```

```
    ncx_nmda_ds_t               nmda_ds;

    /* caller will set the return nmda_origin */
    ncx_nmda_origin_t           nmda_origin;

    /* In AIO GET2 callback is used with JSON/XML buffers
     * the encoding will represent corresponding encoding type
     */
    ncx_msg_encoding_t          aio_encoding;

    /* AIO XML or JSON malloced buffer from callback to use.
     * Must be freed after it is converted to val value.
     */
    xmlChar                     *aio_return_buff;

} getcb_get2_t;
```

The YANG module **yumaworks-sil-sa.yang** defines the get-request and get-response messages.

get-request Message

```
          container get-request {
            description
              "Composite retrieval request to support NETCONF
               and RESTCONF get operations.
               Type: server-request
               Expected Response Message: subsys-response
                                 (get-response or error)";

            uses transaction-id-obj;
            uses user-id-obj;
            uses client-addr-obj;

            leaf flags {
              type bits {
                bit keys {
                  description
                    "Return only the key values";
                }
                bit config {
                  description
                    "Return config=true data nodes";
                }
                bit oper {
                  description
                    "Return config=false data nodes";
                }
                bit getnext {
                  description
                    "This is a get-next request instead of a
                    get request";
                }
                bit withdef {
                  description
                    "Return default values for missing nodes";
                }
                bit select {
                  description
                    "Return only the select nodes and any key leafs.
```

```
          Ignore the config, oper, withdef flags if this
          bit is set.";
      }
      bit with-origin {
        description
          "This is a <get-data> operation and the
           with-origin parameter is selected.";
      }
    }
    default "";
    description
      "Set of get request modifier flags";
  }

  leaf max-entries {
    type uint32;
    description
      "Max number of entries requested.
       The default is '0' == all for leaf-list and
       '1' for all other node types.";
  }

  uses path-parm;

  anyxml keys {
    description
      "List of all ancestor or self key values for the
       object being retrieved, identified by the 'path'
       value. There will be one child leaf for each key
       in each list.";
  }

  anyxml matches {
    description
      "Represents any content-match child leafs for the
       request. All leafs in this container must match the
       corresponding child nodes in an instance of the
       requested list or container, for that instance to
       be returned.

       Any content-match nodes must match in addition
       to any key leafs specified in the 'keys' container.";
  }

  container select-nodes {
    list select-node {
      description
        "Only requesting these nodes be returned. If no
         entries and the 'select' bit is set in the flags
         leaf, then no objects except list keys are
         returned.";

      key objname;
      leaf objname {
        type string;
        description
          "Object name of the select node";
      }
      leaf modname {
        type string;
        description
          "Module name of the select node; If missing then
           use the module-name of the path target object.";
```

```
        }
      }
    }
  }
```

get-response Message

```
  container get-response {
    description
      "Composite retrieval response to support NETCONF
       and RESTCONF get operations.
       Type: subsys-response
       Expected Response Message: none";

    uses transaction-id-obj;

    leaf more-data {
      type boolean;
      default false;
      description
        "Indicates if the GET callback has more data to send";
    }

    leaf match-test-done {
      type boolean;
      default false;
      description
        "Indicates if the requested content-match tests have
         be performed. Ignored if the 'matches' parameter
         is missing or empty.";
    }

    leaf active-case-modname {
      type string;
      description
        "Module name of the active case if there is one.
         Only applies if the GET2 object callback is for
         a YANG choice-stmt.";
    }

    leaf active-case {
      type string;
      description
        "Name of the active case if there is one.
         Only applies if the GET2 object callback is for
         a YANG choice-stmt.";

    }

    leaf origin {
      type string;
      description
        "The NMDA origin value for this object.
         Only applies for config=true list and
         presence containers.";
    }

    grouping return-val {
```

```
            anyxml return-keys {
              description
                "List of all ancestor or self key values for the
                 object being retrieved, identified by the 'path'
                 value. There will be one child leaf for each key
                 in each list.";
              }

            anyxml values {
              description
                "Represents the retrieved values, if any.
                 There will be one child node for each returned
                 value.";
              }
          }

          choice return-choice {
            case return-one {
              description
                "For all objects except YANG list, one entry
                 will be returned.  This can also be used
                 for YANG list, except in GETBULK mode.";
              uses return-val;
            }
            case return-getbulk {
              description
                "For YANG list GETBULK mode.  There will be one entry
                 for each list instance that met the search criteria.
                 If the max_entries parameter was greater than zero,
                 there the number of instances of 'entry' should not
                 exceed this value.";
              list entry {
                // no key!!
                uses return-val;
              }
            }
          }
        }
```

7.9.2 GETBULK Support

The **max-entries** field passed in the **<get-request>** indicates the maximum number of getbulk entries that the GET2 callback is allowed to return.

- The GET2 callback is not required to return more than one entry.
- The max-entries field can be ignored by the GET2 callback.
- The **--max-getbulk** CLI parameter can be used to change the **max-entries** value. The default is '10'.
- The **agt_sil_getbulk_max** agt_profile field contains the configured value at run-time. This field is used to set the **max-entries** parameter in the <get-request> message.
- This parameter is only set for YANG list nodes. A leaf-list callback always returns all instances in 1 request. All other data node callback types are expected to return 1 entry.
- If max-entries is set to zero, then the GET2 callback can return as many entries as desired.
- Existing GET2 code will check the max_entries filed and return ERR_NCX_OPERATION_FAILED if it is not set to '1'. **This code needs to be removed in order to use GETBULK**. New SIL code generated with **yangdump-sdk** does not generate this check on max_entries any more.

In order to use the GETBULK API, a GET2 callback MUST add a call to **getcb_finish_getbulk_entry()** function after the entry is completed. The general code for a GET2 callback stays the same, except that the callback can loop through its code to find the next GETNEXT entry multiple times.

The first instance of a list is requested with a GET request, and none of the keys will be present. This is an indication to the list callback that the first entry is being requested. The GET2 callback can return up to **max-entries** getbulk instances. The next request will be a GETNEXT request, starting where the last getbulk entry left off.

Processing List Keys for GETNEXT and GETBULK

There are 2 types of keys passed to a GET2 function in a “u_” callback file for a list:

1. **ancestor keys:** these key values are always set and always fixed
2. **local keys:** these key values are for the target list object.
    1. **key value:** The value of they key in its base type; Only valid if foo_present flag is true
    2. **foo_present:** true if the foo local key leaf value is valid
    3. **foo_fixed:** true if the foo local key leaf value is fixed. This indicates that a content-match filter for that key leaf value is being processed. If true, then the GET2 callback MUST NOT increment the associated key value. Only entries matching the fixed key value are to be returned.

If all of the local keys are set and the associated “foo_fixed” leaf is also set, then the server is requesting a single list entry. Do not return multiple entries in the case, even if max-entries is greater than 1.

7.9.3 Example get2 callbacks

get2-test.yang

```
module get2-test {
    namespace "http://yumaworks.com/ns/get2-test";
    prefix "get2test";

    revision 2015-02-05;

    leaf get2-leaf {
       config false;
       type int32;
    }

    leaf-list get2-leaf-list {
       config false;
       type int32;
    }

    container get2-pcon {
       config false;
       presence "this is a test";
       leaf pcon-leaf { type int32; }
    }

    choice get2-choice {
      config false;
      case A {
        leaf A1 { type string; }
        leaf A2 { type string; }
      }
      container B {
        leaf B1 { type uint8; }
      }
      case C {
        choice C0 {
          default C0.1;
          case C0.1 {
            leaf X {
              type string;
              default "red";
            }
            leaf Y {
              type string;
              default "blue";
            }
          }
          leaf C0.2 { type uint32; }
        }
        container C1 {
          leaf C3 { type int8; }
          leaf C2b {
            type int16;
            default 101;
          }
          container C2a {
            leaf C2a1 { type int32; }
          }
```

```
      }
    }
  }

  list get2-list {
    config false;
    key "D1 D2";
    leaf D1 { type int8; }
    leaf D2 { type int16; }
    leaf D3 { type int32; }
  }

  list config-list {
    key "A B";
    leaf A { type string; }
    leaf B { type int32; }

    list noconfig-list {
      config false;
      key "E1";
      leaf E1 { type uint8; }
      leaf E2 { type boolean; }
      leaf E3 { type empty; }
    }
  }

}
```

Example C file:

Refer to the SIL-SA file **/usr/share/yumapro/src/get2-test/src/get2-test.c**

Refer to the SIL file **/usr/share/yumapro/src/get2-test-local/src/get2-test.c**

GETBULK Example

The following YANG file is used in this example:

```
module t104 {

  namespace "http://netconfcentral.org/ns/t104";
  prefix t104;
  revision "2015-11-04";

  container top-state {
    config false;
    list A {
      key X;
      leaf X { type int32; }
      leaf Y { type int32; }
      leaf Z { type int32; }
    }
  }

}
```

The following SIL code shows how GETBULK might be implemented for the list **/top-state/A.**

```
/********************************************************************
* FUNCTION u_t104_top_state_A_get
*
* Get database object callback for list A
* Path: /top-state/A
* Fill in 'get2cb' response fields
*
* INPUTS:
*     see ncx/getcb.h for details (getcb_fn2_t)
*
* RETURNS:
*     error status
********************************************************************/
status_t u_t104_top_state_A_get (
    getcb_get2_t *get2cb,
    int32 k_top_state_A_X,
    boolean X_fixed,
    boolean X_present)
{
    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);
    }

    obj_template_t *obj = GETCB_GET2_OBJ(get2cb);

    uint32 max_entries = GETCB_GET2_MAX_ENTRIES(get2cb);
    if (max_entries == 0) {
        max_entries = 10;
    }

    obj_template_t *X_obj =
        obj_find_child(obj, NULL, (const xmlChar *)"X");
    obj_template_t *Y_obj =
        obj_find_child(obj, NULL, (const xmlChar *)"Y");
    obj_template_t *Z_obj =
        obj_find_child(obj, NULL, (const xmlChar *)"Z");

    boolean getbulk_mode = !X_fixed;

    /* 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 */

#define MAX_X  42

    int32 X_val = 0;
```

```
    if (X_present) {
        X_val = k_top_state_A_X;
    } else {
        X_val = 1;
    }
    if (getnext) {
        ++X_val;
    }
    if ((X_val == 0) || (X_val > MAX_X)) {
        return ERR_NCX_NO_INSTANCE;
    }

    int32 seed_num = X_val * 100;
    uint32 entry_count = 0;
    boolean done = false;
    while (!done) {

        val_value_t *X_return_val = val_new_value();
        if (X_return_val == NULL) {
            return ERR_INTERNAL_MEM;
        }
        val_init_from_template(X_return_val, X_obj);
        VAL_INT32(X_return_val) = X_val;
        getcb_add_return_key(get2cb, X_return_val);

        if (GETCB_GET2_FIRST_RETURN_KEY(get2cb) == NULL) {
            return ERR_NCX_NO_INSTANCE;
        }

        /* 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;

        /* For GETNEXT, set the more_data flag true if not sure */
        boolean more_data = (X_val < MAX_X);

        /**** SET more_data FLAG ****/

        GETCB_GET2_MORE_DATA(get2cb) = more_data;

        /* 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 */

            /* real code would find the current list entry
             * and then get the Y and Z leafs out of that entry
             * This code just returns random numbers
             */
            val_value_t *childval = NULL;

            /* leaf Y (int32) */
```

```
            if (childobj == Y_obj) {
                childval = val_new_value();
                if (childval == NULL) {
                    return ERR_INTERNAL_MEM;
                }
                val_init_from_template(childval, childobj);
                srand(seed_num++);
                VAL_INT32(childval) = rand();
                getcb_add_return_val(get2cb, childval);
            } else if (childobj == Z_obj) {
                /* leaf Z (int32) */
                childval = val_new_value();
                if (childval == NULL) {
                    return ERR_INTERNAL_MEM;
                }
                val_init_from_template(childval, childobj);
                srand(seed_num++);
                VAL_INT32(childval) = rand();
                getcb_add_return_val(get2cb, childval);
            }
        }

        if (getbulk_mode) {
            getcb_finish_getbulk_entry(get2cb);

            /* make sure both numbers get incremented */
            ++entry_count;
            ++X_val;

            if ((entry_count == max_entries) || (X_val > MAX_X)) {
                done = true;
            }
        } else {
            done = true;
        }
    }

    return NO_ERR;

} /* u_t104_top_state_A_get */
```

7.9.4 All In One GET2 Callback

This section describes All In One (AIO) GET2 mechanism that allows to call a single GET2 callback for the whole Subtree.

AIO GET2 callbacks follow exactly the same template as regular GET2 callbacks, they use the same registration API and are getting invoked the same way for SIL and SIL-SA as regular GET2 callbacks. The main difference is that there will not be any callback invocations and there will not be any callbacks at all for any of **AIO** node children.

All In One GET2 support:

- Supported only for container and list nodes
- YANG extension must be specified for the top AIO node
- Supported for SIL and SIL-SA
- Supported for XPath and Subtree filtering
- Supported for RESTCONF retrieval
- Supported for <get-bulk> operation
- Supported for NMDA <get-data> operation for "operational" datastore
- Auto-generation code support
- Must be built with **--sil-get2** flag during **make_sil_dir_pro**

The entire subtree would be expected in one retrieval in one callback invocation. That is if the callback is registered for a container, any complex nodes within this container will not have any extra callbacks, they will be handled by the top **AIO** container callback.

With help of a new YANG extension, the server will set specific flags to the objects to identify them as a special case retrieval. This extension would be used in a container or a list to force the server to treat that data subtree as an **AIO** node for GET2.

The **yangdump** auto-generation code does not auto generate callbacks for children of the top All In One parent. The callbacks will be generated only for the top nodes that have the extension specified. The extension definition looks as follows:

```
extension sil-aio-get2 {
  description
    "Used within a data definition statement to define
     the GET2 retrieval mechanism.
     This extension affects the descendant data nodes.

     This extension can be used in a container or list
     to force the server to treat that data subtree as
     a terminal node for GET2.

     The entire subtree would be expected in one retrieval
     in one callback invocation.

     The entire subtree can be specified in the JSON
     or XML buffer that will be used for return values.
     The server will parse and handle the buffer and process
     retrieval based on the provide JSON or XML encoded buffer.

     The 'parmstr' argument can specify the encoding that will
     be used in the callback. Available options are:
       - xml: XML element in a buffer is expected in return value
       - json: JSON object in a buffer is expected in return value
       - val: val_value_t tree is expected in return value
    ";
  argument parmstr;
```

```
  }
```

As an example, to illustrate how the extension can be used in a YANG module refer to the following snippet from YANG module:

get3-test.yang

```
/* All in One callback within config true and GET2 reg callbacks */
  container get3-config-cont {
    presence "Presence container";

    list config-list {
      key name;
      leaf name { type string; }

      list nonconfig-list {              // Regular GET2 CB
        config false;
        key name;
        leaf name { type string; }

        list nonconfig-list2 {           // All in One GET2 CB
          ywx:sil-aio-get2 "val";

          key name;
          leaf name { type string; }

          list nonconfig-list3 {         // No callback
            key name;
            leaf name { type string; }

            container int-con {          // No callback
              leaf int-con-leaf { type int32; }
            }
          }
        }
      }
    }
  }
```

Example C file:

Refer to the SIL-SA file **/usr/share/yumapro/src/get3-test/src/get3-test.c**

Refer to the SIL file **/usr/share/yumapro/src/get3-test-local/src/get3-test.c**

In the YANG module example above we define top level configuration parent and next regular GET2 child and then the All In One GET2 list child.

In this case the server will invoke GET2 callbacks only for **"nonconfig-list"** first and then for **AIO "nonconfig-list2"** list and will not invoke any deeper level callbacks. The server will expect that the **AIO** "**nonconfig-list2**" callback will supply the GET2 control block with the sufficient information about **"nonconfig-list2"** node children, complex children as well as terminal nodes.

7.9.5 AIO List Example

This section illustrates how to use All In One GET2 callbacks for list nodes. Assume we have the same data model as described above then the GET2 callback may look as follow:

```
/********************************************************************
* FUNCTION get_test_cont_list_list
*
* Get database object callback for list nonconfig-list2
* Path: /get3-nonconfig-cont/nonconfig-list/nonconfig-list2
* Fill in 'get2cb' response fields
*
* sil-aio-get2 extension enabled for this object;
* No callbacks for children will be called
*
* INPUTS:
*     see ncx/getcb.h for details (getcb_fn2_t)
*
* RETURNS:
*     error status
********************************************************************/
static status_t
    get_test_cont_list_list (ses_cb_t *scb,
                             xml_msg_hdr_t *msg,
                             getcb_get2_t *get2cb)
{
    obj_template_t *obj = GETCB_GET2_OBJ(get2cb);
    status_t res = NO_ERR;

    /* create 3 top level lists */
    uint32 key = 0;
    for (key = 1; key < 4; key++) {

        const xmlChar *key1 = NULL;
        if (key == (uint32)1) {
            key1 = (const xmlChar *)"name1";
        } else if (key == (uint32)2) {
            key1 = (const xmlChar *)"name2";
        } else if (key == (uint32)3) {
            key1 = (const xmlChar *)"name3";
        }

        /* make return value that will be added to the return_aioQ */
        val_value_t *retval = val_new_value();
        if (!retval) {
            return ERR_INTERNAL_MEM;
        }
        val_init_from_template(retval, obj);

        /* Use retval for the current callback obj and fill
         * it in with folowing terminal and complex nodes;
         * then add to get2cb with help of getcb_add_return_aioQ API
         */
        obj_template_t *childobj =
            getcb_first_requested_child(get2cb, obj);
        for (; childobj; childobj =
            getcb_next_requested_child(get2cb, childobj)) {

            const xmlChar *name = obj_get_name(childobj);
```

```
        /* Retrieve the value of this child node and
         * add it to retval of the current callback node
         */
        if (!xml_strcmp(name, (const xmlChar *)"name")) {
            /* leaf name (string) */
            val_value_t *child =
                val_make_simval_obj(childobj,
                                    key1,
                                    &res);
            if (child) {
                res = val_child_add(child, retval);
                if (res != NO_ERR) {
                    val_free_value(child);
                    val_free_value(retval);
                    return ERR_NCX_INVALID_VALUE;
                }
            } else {
                val_free_value(retval);
                return ERR_NCX_INVALID_VALUE;
            }
        } else if (!xml_strcmp(name, (const xmlChar *)"nonconfig-list3")) {

            /* list nonconfig-list (list) */
            uint32 nextlist_key = 0;
            for (nextlist_key = 1; nextlist_key < 3; nextlist_key++) {

                const xmlChar *keyname = NULL;
                if (nextlist_key == (uint32)1) {
                    keyname = (const xmlChar *)"name1";
                } else if (nextlist_key == (uint32)2) {
                    keyname = (const xmlChar *)"name2";
                }

                /* create 5 list entries */
                val_value_t *listval =
                    create_list_entry(childobj,
                                      (const xmlChar *)"name",
                                      keyname,
                                      &res);
                if (!listval) {
                    val_free_value(retval);
                    return ERR_INTERNAL_MEM;
                }

                /* Use add_force to allow insertion lists with NO keys */
                res = val_child_add(listval, retval);
                if (res != NO_ERR) {
                    val_free_value(listval);
                    val_free_value(retval);
                    return ERR_NCX_INVALID_VALUE;
                }
            }
        }
    }

    if (res == NO_ERR) {
        /* generate the internal index Q chain */
        res = val_gen_index_chain(obj, retval);
        if (res != NO_ERR) {
            val_free_value(retval);
            return res;
        }
```

```
        }

        /* Ensure that the generated value is correct and can be added
         * to the return Queue.
         */
        if (retval && res == NO_ERR) {
            res = val_validate_value(retval);
            if (res != NO_ERR) {
                val_free_value(retval);
                return res;
            }
        }

        /* Add created val_value_t to get2cb with help of
         * getcb_add_return_aioQ API; In case of list callbacks,
         * repeat above steps for another list entry
         */
        getcb_add_return_aioQ(get2cb, retval);
    }

    return res;

} /* get_test_cont_list_list */
```

In the examples above we used a function "create_list_entry" that is a custom function that can be used as an example function that helps to build the subtree data. This is not a server API and used in this article just to demonstrate how the children nodes can be constructed. For more example you can refer to following articles:

Below is the example function that was used in he above **AIO** callback example:

```
/*********************************************************************
* FUNCTION create_list_entry
*
* Make a list entry based on the key
*
* INPUTS:
*   res = return status
*
* RETURNS:
*    val_value_t of listval entry if no error
*    else NULL
*
*********************************************************************/
static val_value_t *
    create_list_entry (obj_template_t *list_obj,
                       const xmlChar *keyname,
                       const xmlChar *keystr,
                       status_t *res)
{
    /* find a key child obejct */
    obj_template_t *key_obj =
        obj_find_child(list_obj,
                       GET3_TEST_MOD,
                       keyname);
    if (!key_obj) {
        *res = ERR_NCX_INVALID_VALUE;
        return NULL;
    }
```

```
    val_value_t *list_value = val_new_value();
    if (!list_value) {
        *res = ERR_NCX_INVALID_VALUE;
        return NULL;
    }
    val_init_from_template(list_value, list_obj);

    /* make key leaf entry */
    val_value_t *child =
        val_make_simval_obj(key_obj,
                            keystr,
                            res);
    if (!child) {
        val_free_value(list_value);
        *res = ERR_NCX_INVALID_VALUE;
        return NULL;
    }

    *res = val_child_add(child, list_value);
    if (*res != NO_ERR) {
        val_free_value(child);
        val_free_value(list_value);
        return NULL;
    }

    obj_template_t *notkey_obj =
        obj_find_child(list_obj,
                       GET3_TEST_MOD,
                       (const xmlChar *)"not-keyname");
    if (notkey_obj) {
        /* make NON key leaf entry */
        child =
            val_make_simval_obj(notkey_obj,
                                (const xmlChar *)"some-value",
                                res);
        if (!child) {
            *res = ERR_NCX_INVALID_VALUE;
        }

        if (*res == NO_ERR) {
            *res = val_child_add(child, list_value);
            if (*res != NO_ERR) {
                val_free_value(child);
            }
        }
    }

    obj_template_t *cont_obj =
        obj_find_child(list_obj,
                       GET3_TEST_MOD,
                       (const xmlChar *)"int-con");
    if (cont_obj) {
        /* make return value that will be added to the return_aioQ */
        val_value_t *contval = val_new_value();
        if (!contval) {
            val_free_value(list_value);
            *res = ERR_NCX_INVALID_VALUE;
            return NULL;
        }
        val_init_from_template(contval, cont_obj);

        obj_template_t *leaf_obj =
            obj_find_child(cont_obj,
```

```
                          GET3_TEST_MOD,
                          (const xmlChar *)"con-leaf");
        if (leaf_obj) {
            val_value_t *leafval =
                val_make_simval_obj(leaf_obj,
                                    (const xmlChar *)"42",
                                    res);
            if (leafval) {
                *res = val_child_add(leafval, contval);
                if (*res != NO_ERR) {
                    val_free_value(leafval);
                    val_free_value(contval);
                }
            } else {
                val_free_value(contval);
                *res = ERR_NCX_INVALID_VALUE;
            }

            if (*res == NO_ERR) {
                *res = val_child_add(contval, list_value);
                if (*res != NO_ERR) {
                    val_free_value(contval);
                }
            }
        } else {
            val_free_value(contval);
        }
    }

    /* generate the internal index Q chain */
    *res = val_gen_index_chain(list_obj, list_value);
    if (*res != NO_ERR) {
        log_error("\nError: could not generate index chain (%s)",
                  get_error_string(*res));
        val_free_value(list_value);
        return NULL;
    }

    return list_value;

} /* create_list_entry */
```

The following section in the **AIO** callback function example is used to ensure that the constructed value is well-formed and can be safely used by the server. It runs simple sanity check to verify that if the value is a list it has all the keys setup. It checks descendant-or-self nodes for lists. Checks if they have index chains built already. If not, then try to add one for each of the key objects in order.

If the value is not a container or a list or if the list value index chain is malformed this function will return an error "invalid value".

```
    /* Ensure that the generated value is correct and can be added
     * to the return Queue.
     */
    if (retval && res == NO_ERR) {
        res = val_validate_value(retval);
        if (res != NO_ERR) {
            val_free_value(retval);
            return res;
```

```
        }
    }
```

The below section of the **AIO** callback is used to add a new val value into the Queue that will be used as a return values of the callbacks. That will be used later for processing, for filtering, for other manipulations:

```
    /* Add created val_value_t to get2cb with help of
     * getcb_add_return_aioQ API; In case of list callbacks,
     * repeat above steps for another list entry
     */
    getcb_add_return_aioQ(get2cb, retval);
```

7.9.6 AIO Container Example

This section illustrates how to use All In One GET2 callbacks for container nodes. Assume we have the following data model:

```
/* All in One callback within config true NP-container */
container get3-config-cont {

  /* NP-container with a terminal node */
  container nonconfig-NPcont {        // All in One get2 CB
    ywx:sil-aio-get2 "val";
    config false;

    list nonconfig-list {             // No callback
      key name;
      leaf name { type string; }
    }
    leaf D { type int8; }
  }
}
```

Then the **AIO** GET2 callback may look as follow:

```
/********************************************************************
* FUNCTION get_test_config_cont_nonconfig_NPcont
*
* Get database object callback for container nonconfig-NPcont
* Path: /get3-config-cont/nonconfig-NPcont
* Fill in 'get2cb' response fields
*
* sil-aio-get2 extension enabled for this object;
* No callbacks for children will be called
*
* INPUTS:
*     see ncx/getcb.h for details (getcb_fn2_t)
*
```

```
* RETURNS:
*     error status
*********************************************************************/
static status_t
    get_test_config_cont_nonconfig_NPcont (ses_cb_t *scb,
                                           xml_msg_hdr_t *msg,
                                           getcb_get2_t *get2cb)
{
    /* an NP container always exists so no test for node_exists
     * by the SIL or SIL-SA callback is needed
     */
    obj_template_t *obj = GETCB_GET2_OBJ(get2cb);
    status_t res = NO_ERR;

    /* make return value that will be added to the return_aioQ */
    val_value_t *retval = val_new_value();
    if (!retval) {
        return ERR_INTERNAL_MEM;
    }
    val_init_from_template(retval, obj);

    /* Use retval for the current callback obj and fill
     * it in with folowing terminal and complex nodes;
     * then add to get2cb with help of getcb_add_return_aioQ API
     */
    obj_template_t *childobj =
        getcb_first_requested_child(get2cb, obj);
    for (; childobj; childobj =
        getcb_next_requested_child(get2cb, childobj)) {

        const xmlChar *name = obj_get_name(childobj);

        /* Retrieve the value of this child node and
         * add it to retval of the current callback node
         */
        if (!xml_strcmp(name, (const xmlChar *)"nonconfig-list")) {
            /* list nonconfig-list (list) */

            uint32 key = 0;
            for (key = 1; key < 3; key++) {

                /* create 2 list entries */
                const xmlChar *keyname = NULL;
                if (key == (uint32)1) {
                    keyname = (const xmlChar *)"name1";
                } else if (key == (uint32)2) {
                    keyname = (const xmlChar *)"name2";
                }

                val_value_t *listval =
                    create_list_entry(childobj,
                                      (const xmlChar *)"name",
                                      keyname,
                                      &res);
                if (!listval) {
                    val_free_value(retval);
                    return ERR_INTERNAL_MEM;
                }

                /* Use add_force to allow insertion lists with NO keys */
                res = val_child_add(listval, retval);
                if (res != NO_ERR) {
                    val_free_value(listval);
```

```
                    val_free_value(retval);
                    return ERR_NCX_INVALID_VALUE;
                }
            }
        } else if (!xml_strcmp(name, (const xmlChar *)"D")) {
            /* leaf D (int8) */
            val_value_t *child =
                val_make_simval_obj(childobj,
                                    (const xmlChar *)"42",
                                    &res);
            if (child) {
                res = val_child_add(child, retval);
                if (res != NO_ERR) {
                    val_free_value(child);
                    val_free_value(retval);
                    return ERR_NCX_INVALID_VALUE;
                }
            } else {
                val_free_value(retval);
                return ERR_NCX_INVALID_VALUE;
            }
        }
    }

    /* Ensure that the generated value is correct and can be added
     * to the return Queue.
     */
    if (retval && res == NO_ERR) {
        res = val_validate_value(retval);
        if (res != NO_ERR) {
            val_free_value(retval);
            return res;
        }
    }

    /* Add created val_value_t to get2cb with help of
     * getcb_add_return_aioQ API; In case of list callbacks,
     * repeat above steps for another list entry
     */
    getcb_add_return_aioQ(get2cb, retval);

    return res;

} /* get_test_config_cont_nonconfig_NPcont */
```

In the examples above we used a function **"create_list_entry"** that is a custom function that can be used as an example function that helps to build the subtree data. This is not a server API and used in this article just to demonstrate how the children nodes can be constructed. For more example you can refer to following articles:

Below is the example function that was used in he above **AIO** callback example:

```
/********************************************************************
* FUNCTION create_list_entry
*
* Make a list entry based on the key
*
* INPUTS:
*   res = return status
*
```

```
* RETURNS:
*    val_value_t of listval entry if no error
*    else NULL
*
*********************************************************************/
static val_value_t *
    create_list_entry (obj_template_t *list_obj,
                       const xmlChar *keyname,
                       const xmlChar *keystr,
                       status_t *res)
{
    /* find a key child obejct */
    obj_template_t *key_obj =
        obj_find_child(list_obj,
                       GET3_TEST_MOD,
                       keyname);
    if (!key_obj) {
        *res = ERR_NCX_INVALID_VALUE;
        return NULL;
    }

    val_value_t *list_value = val_new_value();
    if (!list_value) {
        *res = ERR_NCX_INVALID_VALUE;
        return NULL;
    }
    val_init_from_template(list_value, list_obj);

    /* make key leaf entry */
    val_value_t *child =
        val_make_simval_obj(key_obj,
                            keystr,
                            res);
    if (!child) {
        val_free_value(list_value);
        *res = ERR_NCX_INVALID_VALUE;
        return NULL;
    }

    *res = val_child_add(child, list_value);
    if (*res != NO_ERR) {
        val_free_value(child);
        val_free_value(list_value);
        return NULL;
    }

    obj_template_t *notkey_obj =
        obj_find_child(list_obj,
                       GET3_TEST_MOD,
                       (const xmlChar *)"not-keyname");
    if (notkey_obj) {
        /* make NON key leaf entry */
        child =
            val_make_simval_obj(notkey_obj,
                                (const xmlChar *)"some-value",
                                res);
        if (!child) {
            *res = ERR_NCX_INVALID_VALUE;
        }

        if (*res == NO_ERR) {
            *res = val_child_add(child, list_value);
            if (*res != NO_ERR) {
```

```
                val_free_value(child);
            }
        }
    }

    obj_template_t *cont_obj =
        obj_find_child(list_obj,
                       GET3_TEST_MOD,
                       (const xmlChar *)"int-con");
    if (cont_obj) {
        /* make return value that will be added to the return_aioQ */
        val_value_t *contval = val_new_value();
        if (!contval) {
            val_free_value(list_value);
            *res = ERR_NCX_INVALID_VALUE;
            return NULL;
        }
        val_init_from_template(contval, cont_obj);

        obj_template_t *leaf_obj =
            obj_find_child(cont_obj,
                           GET3_TEST_MOD,
                           (const xmlChar *)"con-leaf");
        if (leaf_obj) {
            val_value_t *leafval =
                val_make_simval_obj(leaf_obj,
                                    (const xmlChar *)"42",
                                    res);
            if (leafval) {
                *res = val_child_add(leafval, contval);
                if (*res != NO_ERR) {
                    val_free_value(leafval);
                    val_free_value(contval);
                }
            } else {
                val_free_value(contval);
                *res = ERR_NCX_INVALID_VALUE;
            }

            if (*res == NO_ERR) {
                *res = val_child_add(contval, list_value);
                if (*res != NO_ERR) {
                    val_free_value(contval);
                }
            }
        } else {
            val_free_value(contval);
        }
    }

    /* generate the internal index Q chain */
    *res = val_gen_index_chain(list_obj, list_value);
    if (*res != NO_ERR) {
        log_error("\nError: could not generate index chain (%s)",
                  get_error_string(*res));
        val_free_value(list_value);
        return NULL;
    }

    return list_value;

} /* create_list_entry */
```

The following section in the **AIO** callback function example is used to ensure that the constructed value is well-formed and can be safely used by the server. It runs simple sanity check to verify that if the value is a list it has all the keys setup. It checks descendant-or-self nodes for lists. Checks if they have index chains built already. If not, then try to add one for each of the key objects in order.

If the value is not a container or a list or if the list value index chain is malformed this function will return an error **"invalid value"**.

```
/* Ensure that the generated value is correct and can be added
 * to the return Queue.
 */
if (retval && res == NO_ERR) {
    res = val_validate_value(retval);
    if (res != NO_ERR) {
        val_free_value(retval);
        return res;
    }
}
```

The below section of the **AIO** callback is used to add a new val value into the Queue that will be used as a return values of the callbacks. That will be used later for processing, for filtering, for other manipulations:

```
/* Add created val_value_t to get2cb with help of
 * getcb_add_return_aioQ API; In case of list callbacks,
 * repeat above steps for another list entry
 */
getcb_add_return_aioQ(get2cb, retval);
```

7.9.7 All In One GET2 Callbacks with XML/JSON buffers

AIO GET2 callbacks follow exactly the same template as regular GET2 callbacks, they use the same registration API and are getting invoked the same way for SIL and SIL-SA as regular GET2 callbacks. The main difference is that there will not be any callback invocations for descendant children and there will not be any callbacks auto generated at all for any of **AIO** node children.

Let us go through simple examples that will illustrate how to utilize the **AIO** callbacks for the specific purposes. First we need a YANG module. Consider this simplified, but functional, example. You can download this YANG module from attachments and run **make_sil_dir_pro** to auto-generate stub SIL code for this module.

Note, the **AIO** callback is not part of the auto-generated code by default and you will have to add an extension to the desired node that you want to become an **AIO** callback node. Otherwise, the auto generated code will generate regular GET2 callback for that node(s).

In this example we are going to use "sil-aio-get2" extension with 'xml' and "json" parameters, meaning we are going to use XML and JSON buffers as a return values in the **AIO** callback.

As an example, to illustrate how the "sil-aio-get2" extension can be used in a YANG module refer to the following YANG module:

```
module aio-test {
  namespace "http://yumaworks.com/ns/aio-test";
  prefix "aiotest";

  import yumaworks-extensions { prefix ywx; }

  revision 2020-02-20 {
    description "Initial Version";
  }


  /* All in One callback within config true and GET2 reg callbacks */
  container get3-config-cont {
    presence "Presence container";

    list config-list {
      key name;
      leaf name { type string; }

      list nonconfig-list {              // Regular GET2 CB
        config false;
        key name;
        leaf name { type string; }

        list nonconfig-list2 {           // All in One XML GET2 CB
          ywx:sil-aio-get2 "xml";

          key name;
          leaf name { type string; }

          list nonconfig-list3 {         // No callback
            key name;
            leaf name { type string; }
            leaf not-keyname { type string; }

            container int-con {          // No callback
              leaf con-leaf { type int32; }
            }
```

```
        }
      }
    }
  }
}

/* All in One callback within config true NP-container */
container get3-state-cont {          // All in One get2 CB JSON
  ywx:sil-aio-get2 "json";
  config false;

  leaf D { type int8; }

  choice thing-choice {               // No callback
    case A {
      leaf A1 { type string; }
      leaf A2 { type string; }
    }
    case B {
      leaf B3 { type string; }
      leaf B2 { type string; }
    }
    case C {                          // active case
      leaf C1 {type string; }
      leaf C2 {type string; }
    }
  }
}
}
```

AIO XML Callback example

In the YANG module example above we define top level configuration parent "**get3-config-cont/config-list**" and next regular GET2 child "**nonconfig-list**" and then the **AIO** GET2 list child "**nonconfig-list2**" with XML parameter in the extension.

In this case the server will invoke GET2 callbacks only for **"nonconfig-list"** and then for **AIO "nonconfig-list2"** list and will not invoke any descendant level callbacks. The server will expect that the **AIO "nonconfig-list2"** callback will supply well formed XML buffer to the GET2 control block.

NOTE: if the callback target is a list, the XML buffer must provide parent node in the buffer. If the list is a top level node use the following wrapper:

```
     <config xmlns=\"http://netconfcentral.org/ns/yuma-ncx\"></config>
```

There cannot be multiple siblings in XML, it will make XML malformed and the server will not be able to parse this buffer and will return an error. As a result the server will skip over this callback nodes in the output and will not generate any output for them.

The **AIO** GET2 callback may look as follow for the "**/get3-config-cont/config-list/nonconfig-list/nonconfig-list2**" list.

```
/*********************************************************************
* FUNCTION aio_test_get3_config_cont_config_list_nonconfig_list_nonconfig_list2_get
*
* Get database object callback for list nonconfig-list2
* Path: list /get3-config-cont/config-list/nonconfig-list/nonconfig-list2
* Fill in 'get2cb' response fields
*
* sil-aio-get2 extension enabled for this object;
* No callbacks for children will be called
*
* XML encoded buffer is expected
*
* INPUTS:
*     see ncx/getcb.h for details (getcb_fn2_t)
*
* RETURNS:
*     error status
*********************************************************************/
static status_t
    aio_test_get3_config_cont_config_list_nonconfig_list_nonconfig_list2_get (
        ses_cb_t *scb,
        xml_msg_hdr_t *msg,
        getcb_get2_t *get2cb)
{
    if (LOGDEBUG) {
        log_debug("\nEnter
aio_test_get3_config_cont_config_list_nonconfig_list_nonconfig_list2_get");
    }

    (void)scb;
    (void)msg;

    status_t res = NO_ERR;

    /* make XML buffer that will be used instead of return val value
     * NOTE if the callback target is a list, the XML buffer must provide
     * parent node in the buffer. There cannot be multiple siblings
     * in XML, it will make XML malformed.
     *
     * If the list is a top level node use the following wrapper:
     * <config xmlns=\"http://netconfcentral.org/ns/yuma-ncx\"></config>
     */
    const xmlChar *xml_buffer = (const xmlChar *)
        "<nonconfig-list xmlns=\"http://yumaworks.com/ns/aio-test\">"
        "<nonconfig-list2>"
        "<name>name1</name><nonconfig-list3>"
        "<name>name1</name></nonconfig-list3><nonconfig-list3>"
        "<name>name2</name></nonconfig-list3></nonconfig-list2>"
        "<nonconfig-list2>"
        "<name>name2</name><nonconfig-list3>"
        "<name>name1</name></nonconfig-list3><nonconfig-list3>"
        "<name>name2</name></nonconfig-list3></nonconfig-list2>"
        "<nonconfig-list2>"
        "<name>name3</name><nonconfig-list3>"
        "<name>name1</name></nonconfig-list3><nonconfig-list3>"
        "<name>name2</name></nonconfig-list3></nonconfig-list2>"
        "</nonconfig-list>";

    /* Add created buffer to return get2cb */
    res = getcb_add_return_aio_buff(get2cb, xml_buffer);
```

```
    return res;

} /* aio_test_get3_config_cont_config_list_nonconfig_list_nonconfig_list2_get */
```

AIO JSON Callback Examples

A JSON example will be similar to the above XML example, it will use the same API, however instead of using XML buffer the callback will provide JSON buffer to the server.

Assume the same YANG module example above we define top level **AIO** JSON container **"get3-state-cont"**.

In this case the server will invoke **AIO** JSON callback for **"get3-state-cont"** container and will not invoke any descendant level callbacks. The server will expect that the **AIO** JSON **"get3-state-cont"** callback will supply the GET2 control block with well formed JSON buffer that will be used as a return value for the callback.

The AIO JSON GET2 callback may look as follow for the **"get3-state-cont"** container.

```
/*********************************************************************
* FUNCTION aio_test_get3_state_cont_get
*
* Get database object callback for container get3-state-cont
* Path: container /get3-state-cont
* Fill in 'get2cb' response fields
*
* sil-aio-get2 extension enabled for this object;
* No callbacks for children will be called
*
* JSON encoded buffer is expected
*
* INPUTS:
*     see ncx/getcb.h for details (getcb_fn2_t)
*
* RETURNS:
*     error status
*********************************************************************/
static status_t
    aio_test_get3_state_cont_get (ses_cb_t *scb,
                                  xml_msg_hdr_t *msg,
                                  getcb_get2_t *get2cb)
{
    if (LOGDEBUG) {
        log_debug("\nEnter aio_test_get3_state_cont_get");
    }

    (void)scb;
    (void)msg;

    status_t res = NO_ERR;

    /* make JSON buffer that will be used instead of return val value */
    const xmlChar *json_buffer = (const xmlChar *)
        "{\"aio-test:get3-state-cont\":{\"D\":42,\"C2\":\"someleaf\"}}";;

    /* Add created buffer to return get2cb */
    res = getcb_add_return_aio_buff(get2cb, json_buffer);

    return res;
```

```
} /* aio_test_get3_state_cont_get */
```

As a result, whenever some north bound agent trying to retrieve the list or container that we have **AIO** callbacks registered, the callback is invoked and a client gets all the values that **AIO** callbacks returns.

To ensure that the buffer returned by **AIO** callback was successfully consumed by the server an application can retrieve operational datastore with help of the following operation as an example. Assume, we already created parent configuration nodes.

```
 <get>
   <filter type="subtree">
     <get3-config-cont xmlns="http://yumaworks.com/ns/aio-test"/>
   </filter>
 </get>
```

The server should reply with:

```
<data>
  <get3-config-cont xmlns="http://yumaworks.com/ns/aio-test">
   <config-list>
    <name>list1</name>
    <nonconfig-list>
     <name>name1</name>
     <nonconfig-list2>
      <name>name1</name>
      <nonconfig-list3>
       <name>name1</name>
      </nonconfig-list3>
      <nonconfig-list3>
       <name>name2</name>
      </nonconfig-list3>
     </nonconfig-list2>

       ...

    </nonconfig-list>
   </config-list>
  </get3-config-cont>
 </data>
```

Or, for example, if we want to retrieve a node that was generated by **AIO** JSON callback, a client may send the following <get> request:

```
 <get>
   <filter type="subtree">
     <get3-state-cont xmlns="http://yumaworks.com/ns/aio-test"/>
   </filter>
 </get>
```

The server should reply with:

```
<data>
 <get3-state-cont xmlns="http://yumaworks.com/ns/aio-test">
  <D>42</D>
  <C2>someleaf</C2>
 </get3-state-cont>
</data>
```

7.9.8 Static Operational Data

- The **agt_make_leaf** function or one of its variants is used to create a static object that is added to the parent container or list with the **val_add_child** or **val_add_child_sorted** function. Only use val_add_child when constructing a YANG data structure in the correct order. Always use **val_add_child_sorted** when adding nodes to an existing data structure.
- The value returned for retrieval requests is stored in the val_value_t structure in the running datastore tree.
- Use static operational data when the data to be returned in an <rpc-reply> is static.

Example Static Node Creation

```
/* 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;
}
```

7.9.9 Virtual Operational Data

- The **agt_make_virtual_leaf** function or one of its variants is used to create a dynamic object that is added to the parent container or list with the **val_add_child_sorted** function..
- A pointer to a "**getcb**" template callback function is stored in the val_value_t in the running datastore tree.
- When the value is retrieved the callback function is called to retrieve the current value. This value is cached in the running datastore tree value for a short time period.
- Use virtual operational data when the data is stored in the running system somewhere and the values need to be retrieved from the system in order to fill the YANG data nodes in the <rpc-reply>.

Get Callback Template

```
/* getcb_fn_t
 *
 * Callback function for agent node get handler
 *
 * INPUTS:
 *   scb    == session that issued the get (may be NULL)
 *             can be used for access control purposes
 *   cbmode == reason for the callback
 *   virval == place-holder node in the data model for
 *              this virtual value node
 *   dstval == pointer to value output struct
 *
 * OUTPUTS:
 *  *fil may be adjusted depending on callback reason
 *  *dstval should be filled in, depending on the callback reason
 *
 * RETURNS:
 *    status:
 */
typedef status_t
    (*getcb_fn_t) (ses_cb_t *scb,
                   getcb_mode_t cbmode,
                   const val_value_t *virval,
                   val_value_t *dstval);
```

Get Callback Example

This example shows a callback for the **/system/sysCurrentDateTime** callback function.

The SIL code adds a child leaf to the "system" parent container using the **agt_make_virtual_leaf** function:

```
    /* 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;
    }
```

The **get_currentDateTime** function that was registered above is used to retrieve the system time when the function is called.

```
/********************************************************************
* FUNCTION get_currentDateTime
*
* <get> operation handler for the sysCurrentDateTime leaf
*
* INPUTS:
*    see ncx/getcb.h getcb_fn_t for details
*
* RETURNS:
*    status
********************************************************************/
static status_t
    get_currentDateTime (ses_cb_t *scb,
                         getcb_mode_t cbmode,
                         const val_value_t *virval,
                         val_value_t  *dstval)
{
    xmlChar      *buff;

    (void)scb;
    (void)virval;

    if (cbmode == GETCB_GET_VALUE) {
        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 */
```

7.10 Periodic Timer Service

Some SIL code may need to be called at periodic intervals to check system status, update counters, and/or perhaps send notifications.

The file **agt/agt_timer.h** contains the timer access function declarations.

This section provides a brief overview of the SIL timer service.

7.10.1 Timer Callback Function

The timer callback function is expected to do a short amount of work, and not block the running process. The function returns zero for a normal exit, and -1 if there was an error and the timer should be destroyed.

The **agt_timer_fn_t** template in **agt/agt_timer.h** is used to define the SIL timer callback function prototype. This typedef defines the callback function template expected by the server for use with the timer service:

```
/* timer callback function
 *
 * Process the timer expired event
 *
 * INPUTS:
 *    timer_id == timer identifier
 *    cookie == context pointer, such as a session control block,
 *            passed to agt_timer_set function (may be NULL)
 *
 * RETURNS:
 *     0 == normal exit
 *    -1 == error exit, delete timer upon return
 */
typedef int (*agt_timer_fn_t) (uint32  timer_id,
                               void *cookie);
```

The following table describes the parameters for this callback function:

SIL Timer Callback Function Parameters

Parameter	Description
timer_id	The timer ID that was returned when the **agt_timer_create** function was called.
cookie	The cookie parameter value that was passed to the server when the **agt_timer_create** function was called.

7.10.2 Timer Access Functions

A SIL timer can be set up as a periodic timer or a one-time event.

The timer interval (in seconds) and the SIL timer callback function are provided when the timer is created. A timer can also be restarted if it is running, and the time interval can be changed as well.

The following table highlights the SIL timer access functions in **agt/agt_timer.h:**

SIL Timer Access Functions

Function	Description
agt_timer_create	Create a SIL timer.
agt_timer_restart	Restart a SIL timer
agt_timer_delete	Delete a SIL timer

7.10.3 Example Timer Callback Function

The following example from **toaster.c** simply completes the toast when the timer expires and calls the auto-generated '**toastDone**' send notification function:

```
/********************************************************************
* FUNCTION toaster_timer_fn
*
* Added timeout function for toaster function
*
* INPUTS:
*     see agt/agt_timer.h
*
* RETURNS:
*     0 for OK; -1 to kill periodic timer
********************************************************************/
static int
    toaster_timer_fn (uint32 timer_id,
                      void *cookie)
{
    (void)timer_id;
    (void)cookie;

    /* toast is finished */
    toaster_toasting = FALSE;
    toaster_timer_id = 0;
    if (LOGDEBUG2) {
        log_debug2("\ntoast is finished");
    }
    y_toaster_toastDone_send((const xmlChar *)"done");
    return 0;

} /* toaster_timer_fn */
```

7.11 Terminal Diagnostic Messages

The server can support generation of notification messages that will automatically be displayed on yangcli-pro and yp-shell terminal sessions. A SIL or SIL-SA program will generate the <term-msg> notifications and the server will deliver them to all clients enabled to receive notifications. The client program will display the text message contents on the terminal.

- The **yumaworks-term-msg** module defines the <term-msg> notification.
- The **--with-term-msg** CLI parameter is used by netconfd-pro and yangcli-pro/yp-shell to enable the <term-msg> notifications
- The **agt_make_term_msg** API function is used to make a notification structure using the provided text string.

```
/*****************************************************************
* FUNCTION agt_make_term_msg
*
* Create a <term-msg> notification
*
* INPUTS:
*    msg == string to send as the data field
*    res == address of return status
* OUTPUTS:
*   *res == return status
* RETURNS:
*    malloced agt_not_msg_t struct if OK; NULL if some error
*****************************************************************/
agt_not_msg_t *
    agt_make_term_msg (const xmlChar *msg,
                       status_t *res);
```

SIL Example:

```
  const xmlChar *msgbuff =
     (const xmlChar *)”This is a 1 line message”;
  status_t res = NO_ERR;
   agt_not_msg_t *notif = agt_make_term_msg(msgbuff, &res);
   if (notif == NULL) {
       log_error("\nError: cannot make term-msg (%s)",
                 get_error_string(res));
       return;
   }

   agt_not_queue_notification(notif);
```

SIL-SA Example:

```
  const xmlChar *msgbuff =
     (const xmlChar *)”This is a 2 line message\nthat will be displayed”;
```

```
  status_t res = NO_ERR;
   agt_not_msg_t *notif = agt_make_term_msg(msgbuff, &res);
   if (notif == NULL) {
       log_error("\nError: cannot make term-msg (%s)",
                 get_error_string(res));
       return;
   }

   sil_sa_queue_notification(notif);
```

8 Network Management Datastore Architecture (NMDA)

The server supports some NMDA functionality. This must be enabled with the **--with-nmda=true** CLI parameter.

If NMDA is enabled then the ietf-netconf-nmda.yang module will be loaded and the <get-data> operation will be available.

8.1 NMDA Support

The following components of NMDA are supported by the server:

- RFC 8342: NMDA
  - ietf-datastores module
  - ietf-origin module
    -
- RFC 8526: NETCONF Extensions for the NMDA
  - <get-data> operation

The following components of NMDA are not supported:

- RESTCONF protocol
  - Use the operation resource to access the <get-data> operation
- NETCONF <edit-data> operation
  - No useful functionality unless and until dynamic datastores supported
- RFC 8525: YANG Library
  - No useful functionality unless and until dynamic datastores supported

8.2 Instrumentation Changes for NMDA

The main difference for SIL code developers is that the server now supports GET2 callback functions for **config=true** data nodes. The operational values of configuration data nodes must be provided by SIL or SIL-SA GET2 callback functions.

If the **--sil-nmda** and **--sil-get2** flags are used with any of the **make_sil_dir** scripts, then **yangdump-sdk** will generate code for NMDA. This consists of GET2 callbacks for the **config=true** nodes, register callbacks, and unregister callbacks for these new callbacks.

There are only 2 minor differences between a GET2 callback for the operational values of configuration data and regular GET2 callbacks:

1. **datastore parameter**: The NMDA datastore being accessed is provided within the GET2 control block parameter (get2cb). Only the <operational> datastore will be accessed using GET2 but in the future other datastores will be able to use GET2 callbacks as well.
2. **origin return value:** The GET2 callback is expected to set the NMDA origin property for the returned data. This can be set for a P-container, list, leaf, or leaf-list. The server will use this meta-data to support the <get-data> operation.

The return origin value is set 2 ways:

- GETCB_GET2_ORIGIN macro to set origin for the list or P-container
- **val_set_origin** function used to set the origin for a child node returned with **getcb_add_return_val** function.

8.3 Example NMDA GET2 Function

```
/*********************************************************************
* FUNCTION u_taddress_addresses_address_get
*
* Get database object callback for list address
* NMDA GET2 for Operational Data
* Path: /addresses/address
* Fill in 'get2cb' response fields
*
* INPUTS:
*     see ncx/getcb.h for details (getcb_fn2_t)
*
* RETURNS:
*     error status
*********************************************************************/
status_t u_taddress_addresses_address_get (
    getcb_get2_t *get2cb,
    const xmlChar *k_addresses_address_last_name,
    boolean last_name_fixed,
    boolean last_name_present,
    const xmlChar *k_addresses_address_first_name,
    boolean first_name_fixed,
    boolean first_name_present)
{

    if (LOGDEBUG) {
        log_debug("\nEnter u_taddress_addresses_address_get");
    }

    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);
    }


    /* TBD: GET2 callbacks for any datastore; expect operational */
    ncx_nmda_ds_t datastore = GETCB_GET2_DATASTORE(get2cb);
    (void)datastore;  // REMOVE THIS LINE IF datastore USED

    obj_template_t *obj = GETCB_GET2_OBJ(get2cb);
    status_t res = NO_ERR;
```

```
uint32 max_entries = GETCB_GET2_MAX_ENTRIES(get2cb);
(void)max_entries;  // REMOVE THIS LINE IF max_entries USED


/* 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 ****/

int retidx = -1;

if (getnext) {
    boolean do_lookup = TRUE;
    if (!last_name_present && !first_name_present) {
        retidx = 0;  // GETNEXT first
        do_lookup = FALSE;
    } else if (last_name_present && first_name_present) {
        if (last_name_fixed && first_name_fixed) {
            // no next entry for fixed keys
            do_lookup = FALSE;
        } // else lookup the next entry of both keys
    } // else lookup the next entry without all keys

    if (do_lookup) {
        retidx = find_next_address(k_addresses_address_last_name,
                                   last_name_fixed,
                                   last_name_present,
                                   k_addresses_address_first_name,
                                   first_name_fixed,
                                   first_name_present);
    }
} else {
    if (!last_name_present && !first_name_present) {
        retidx = 0;  // GETNEXT first
    } else if (last_name_present && first_name_present) {
        retidx = find_exact_address(k_addresses_address_last_name,
                                    k_addresses_address_first_name);
    } // else cannot lookup the exact entry without all keys
}

address_t *a = NULL;
if (retidx >= 0) {
    a = &addresses[retidx];
    val_value_t *keyval =
        agt_make_leaf2(obj,
                       y_taddress_M_taddress,
                       y_taddress_N_last_name,
                       (const xmlChar *)a->last_name,
                       &res);
    if (keyval) {
        getcb_add_return_key(get2cb, keyval);
    } else {
        return res;
    }

    keyval =
        agt_make_leaf2(obj,
                       y_taddress_M_taddress,
                       y_taddress_N_first_name,
```

```
                        (const xmlChar *)a->first_name,
                        &res);
    if (keyval) {
        getcb_add_return_key(get2cb, keyval);
    } else {
        return res;
    }
}

if (GETCB_GET2_FIRST_RETURN_KEY(get2cb) == NULL) {
    return ERR_NCX_NO_INSTANCE;
}

/* Set origin to the correct enumeration found in ncxtypes.h */
ncx_nmda_origin_t origin = NCX_NMDA_ORIGIN_INTENDED;
GETCB_GET2_ORIGIN(get2cb) = origin;

/* 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;

/* For GETNEXT, set the more_data flag to TRUE */
boolean more_data = FALSE;

/**** SET more_data FLAG ****/
if (retidx < MAX_ADDRESS) {
    more_data = TRUE;
}
GETCB_GET2_MORE_DATA(get2cb) = more_data;

/* 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)) {

    const xmlChar *name = obj_get_name(childobj);

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

    val_value_t *retval = NULL;

    if (!xml_strcmp(name, y_taddress_N_street)) {
        /* leaf street (string) */
        retval = val_make_simval_obj(childobj,
                                     (const xmlChar *)a->street,
                                     &res);
    } else if (!xml_strcmp(name, y_taddress_N_city)) {
        /* leaf city (string) */
        retval = val_make_simval_obj(childobj,
                                     (const xmlChar *)a->city,
                                     &res);
    } else if (!xml_strcmp(name, y_taddress_N_zipcode)) {
        /* leaf zipcode (string) */
```

```
            retval = val_make_simval_obj(childobj,
                                         (const xmlChar *)a->zipcode,
                                         &res);
            if (retval) {
                val_set_origin(retval, NCX_NMDA_ORIGIN_LEARNED);
            }
        }
        if (retval) {
            getcb_add_return_val(get2cb, retval);
        } else {
            return res;
        }
    }

    return res;

} /* u_taddress_addresses_address_get */
```

Highlights:

Datastore parameter passed to callback but it can be ignored for now. The value will be ‘operational’

```
    /* GET2 callbacks for any datastore; expect operational */
    ncx_nmda_ds_t datastore = GETCB_GET2_DATASTORE(get2cb);
    (void)datastore;  // REMOVE THIS LINE IF datastore USED
```

Return origin value set for the list entry:

```
    /* Set origin to the correct enumeration found in ncxtypes.h */
    ncx_nmda_origin_t origin = NCX_NMDA_ORIGIN_INTENDED;
    GETCB_GET2_ORIGIN(get2cb) = origin;
```

Return origin value set for a leaf or leaf-list child node returned within a GET2 callback:

```
        val_set_origin(retval, NCX_NMDA_ORIGIN_LEARNED);
```

9 Development Environment

This section describes the YumaPro Tools development environment used to produce the Linux binaries.

9.1 Programs and Libraries Needed

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

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

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

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

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

- **netconf-subsystem-pro (needed by netconfd-pro)**
  - 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-pro** server..
    - After establishing a connection with the **netconfd-pro** server, this program simply transfers SSH2 NETCONF channel data between **sshd** and **netconfd-pro**.

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

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

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

- **ncurses (needed by yangcli-pro)**

    - character processing library needed by **libtecla**; used within **yangcli-pro**
- **libc (or glibc, needed by all applications)**
    - unix system library
- **libssh2 (needed by yangcli-pro)**
    - SSH2 client library, used by yangcli-pro
- **libxml2 (needed by all applications)**
    - xmlTextReader XML parser
    - pattern support

9.2 SIL Shared Libraries

The SIL callback functions are built as shared libraries that are loaded dynamically and accessed by the server with the **dlopen** and **dlsym** library functions. The user callback functions and their source files use the "u_" prefix. YumaPro callback functions and their files use the "y_" prefix.

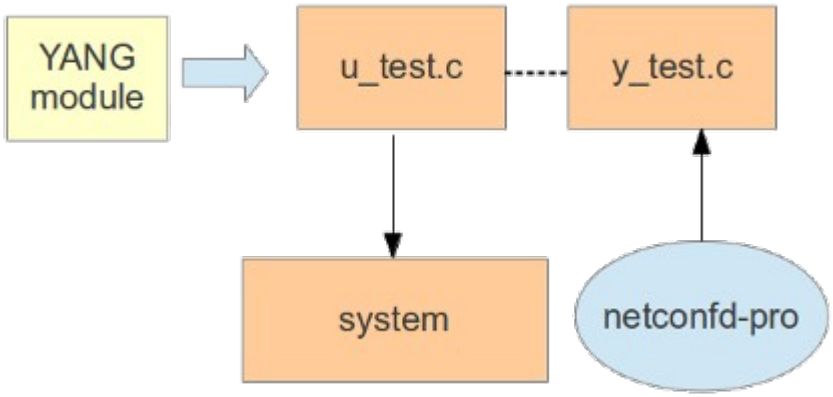
SIL callback code is executed in the netconfd-pro process.

The following YANG statements are supported in SIL code:

- <rpc> statement
- <notification> statement
- Any data definition statement

The following YANG statement is supported in SIL code if built into a SIL bundle

- <augment> statement for external module



The **make_sil_dir_pro** script can be used to generate the C and H file code stubs for a single YANG module. This script uses the y**angdump-sdk** program to generate these files. In this mode, SIL code will be generated only for the definitions in the target module. External augment statements from the target module (or any other module) will not affect SIL code generation.

SIL libraries for single modules are loaded into the server with the **--module** parameter.

The **make_sil_bundle** script can be used to generate the C and H file code stubs for a set of YANG modules. This script uses the y**angdump-sdk** program to generate these files, but with the **--sil-bundle** parameter set.. In this mode, SIL code will be generated for the definitions in the target module. External augment statements from all the modules loaded together will affect SIL code generation.

SIL libraries for SIL bundles are loaded into the server with the **--bundle** parameter.

SIL libraries cannot be generated for YANG submodules. When using **yangdump-sdk** to generate SIL files, use the **--unified=true** option to generate SIL code for the entire module.

9.2.1 SIL Library Names

SIL libraries must be named with a predictable string in order for the server to find it with minimal configuration.

The name must be “lib” + <module-name> + “.” + <file-ext>

where:

- <module-name> is the YANG module name field
- <file-ext> is the shared library file extension (usually “so”)

Example:

```
test.yang → test.c → libtest.so
```

9.2.2 SIL Library Location

By default, SIL binaries are installed in the **/usr/lib/yumapro** directory (**/usr/lib64/yumapro** on Fedora x64 platforms).

The **YUMAPRO_RUNPATH** (or **--runpath** CLI parameter) is used to specify alternate locations for SIL library binary files.

9.2.3 make_sil_dir_pro

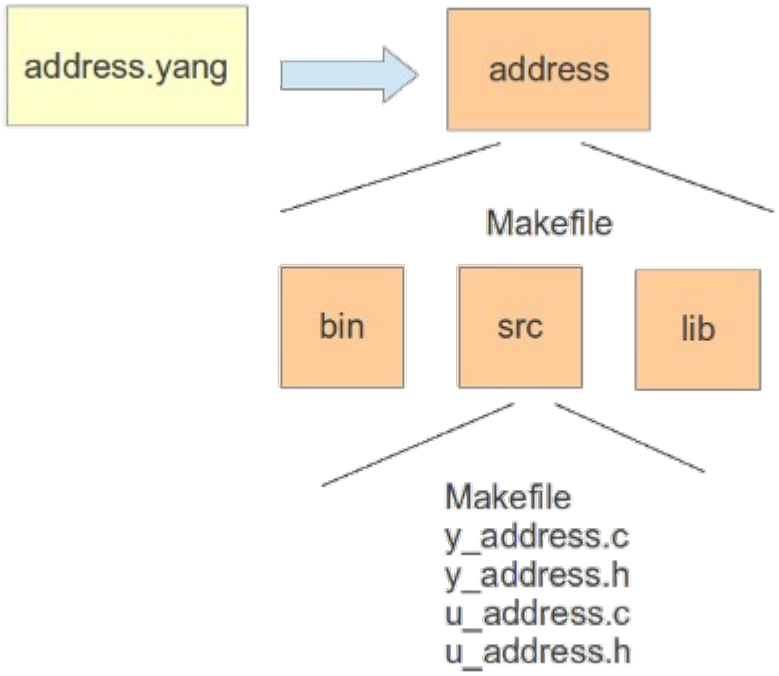
The **make_sil_dir_pro** script should be used to generate SIL code stub files for a single YANG module.

The **--split** parameter should also be used to generate separate user and system code stubs. This is the recommended method in order to minimize coding effort. Combined SIL code stubs (without the **--split** option) can be useful for highly optimized or customized callback design.

The script will generate a directory sub-tree and fill in the Makefiles and source files.

Example:

```
> make_sil_dir_pro --split address
```



Refer to the **make_sil_dir_pro** manual page (“man make_sil_dir_pro”) for more details.

9.2.4 make_sil_bundle

The **make_sil_bundle** script should be used to generate SIL code stub files for multiple YANG modules.

The parameters are a list of strings.

The first parameter is the bundle name. Any additional parameters are module names. These module names should be listed so the base modules are specified first and modules that augment the base modules are listed last.

The script will generate a directory sub-tree and fill in the Makefiles and source files.

Example:

```
> make_sil_bundle ifbundle ietf-interfaces ietf-ip
```

In this example, a bundle named 'ifbundle' is created. The same directory structure is used as for a single module. C files are created for the bundle as well as the module files. The command **--bundle=<bundle-name>** is used to load a SIL bundle into the server at boot-time.

9.2.5 Building SIL Libraries

The compiler flags passed to “make” are extremely important when building SIL source code. The flags must align with those used to build the main server.

Specifically, the “yumapro-pthreads” server must use SIL code that is compiled with the PTHREADS=1 compiler flag. This flag is used in some H files and therefore data structures will be misaligned if the same value is not used in both the server and the SIL libraries.

The version string for a PTHREADS server build contains the phrase “THD” at the end. The command “netconfd-pro --version” can be used to print the server version string.

The DEBUG=1 flag can also be used when building SIL code. This will cause the --ggdb compiler flag to be used.

9.3 SIL-SA Libraries

The SIL-SA callback functions are usually built as shared libraries that are loaded dynamically and accessed by an external application process with the **dlopen** and **dlsym** library functions.

A SIL-SA library can also be built as a static library and linked to the “sil-sa-app” or “combo-app” programs.

The user callback functions and their source files use the “u_” prefix. YumaPro callback functions and their files use the “y_” prefix.

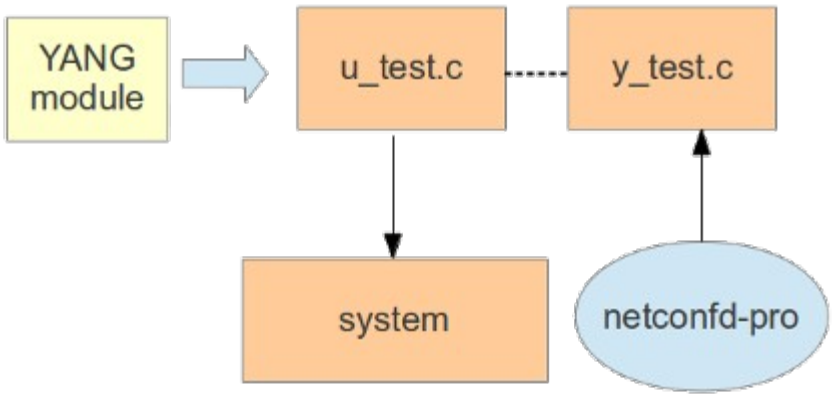
SIL-SA callback code is executed in your external process.

The following YANG statements are supported in SIL-SA code:

- Any data definition statement

The following YANG statements are NOT supported in SIL-SA code at this time. This includes:

- <rpc> statement
- <notification> statement



The **make_sil_sa_dir** script can be used to generate the C and H file code stubs for a single YANG module. This script uses the **yangdump-sdk** program to generate these files. In this mode, SIL code will be generated only for the definitions in the target module. External augment statements from the target module (or any other module) will not affect SIL code generation.

SIL libraries for single modules are loaded into the server with the **--module** parameter.

The **make_sil_sa_bundle** script can be used to generate the C and H file code stubs for a set of YANG modules. This script uses the y**angdump-sdk** program to generate these files, but with the **--sil-bundle** parameter set.. In this mode, SIL code will be generated for the definitions in the target module. External augment statements from all the modules loaded together will affect SIL code generation.

SIL-SA libraries for SIL-SA bundles are loaded into the proper subsystem with the **--bundle** parameter on the server.

SIL-SA libraries cannot be generated for YANG submodules. When using **yangdump-sdk** to generate SIL files, use the **--unified=true** option to generate SIL code for the entire module.

9.3.1 SIL-SA Library Names

SIL libraries must be named with a predictable string in order for the server to find it with minimal configuration.

The name must be “lib” + <module-name> + **“_sa”** + “.” + <file-ext>

where:

- <module-name> is the YANG module name field
- <file-ext> is the shared library file extension (usually “so”)

Example:

```
test.yang → test.c → libtest_sa.so
```

9.3.2 SIL-SA Library Location

By default, SIL-SA binaries are installed in the **/usr/lib/yumapro** directory (**/usr/lib64/yumapro** on Fedora x64 platforms).

The **YUMAPRO_RUNPATH** (or **--runpath** CLI parameter) is used to specify alternate locations for SIL library binary files.

9.3.3 make_sil_sa_dir

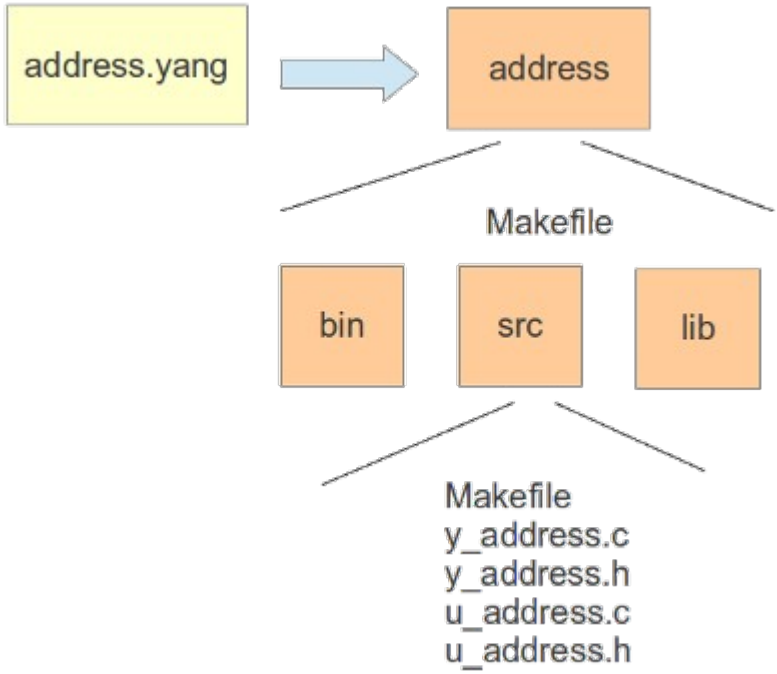
The **make_sil_sa_dir** script should be used to generate SIL-SA code stub files for a single YANG module.

The **--split** parameter should also be used to generate separate user and system code stubs. This is the recommended method in order to minimize coding effort.

The script will generate a directory sub-tree and fill in the Makefiles and source files.

Example:

```
> make_sil_sa_dir --split address
```



Refer to the **make_sil_sa_dir** manual page (“man make_sil_sa_dir”) for more details.

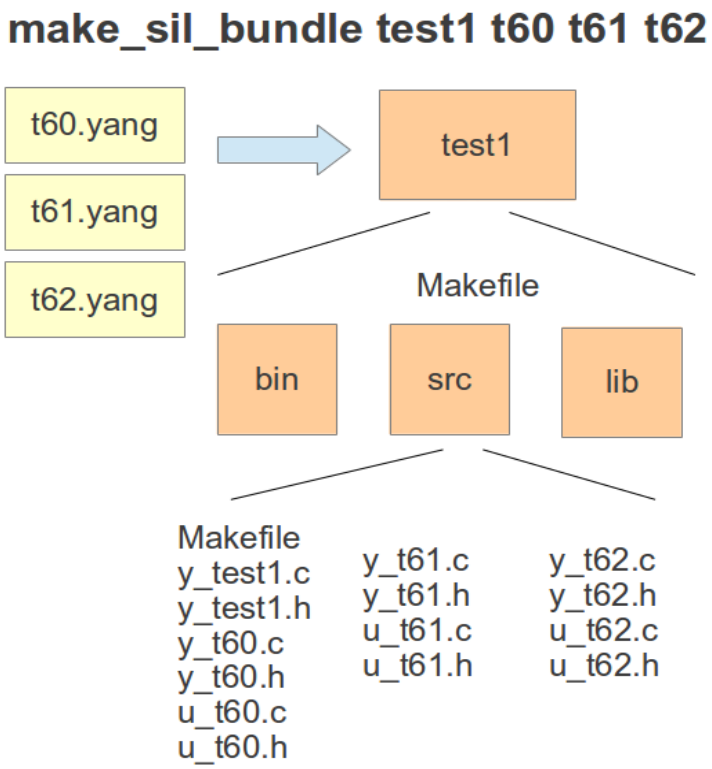
9.4 SIL-SA Bundles

SIL-SA bundles are similar to a SIL-SA for a single YANG module, but a bundle can contain multiple YANG modules. This is useful to simplify development and/or support automatic code stub generation for augment statements that add definitions to another YANG module.

The same YANG usage restrictions for SIL bundles apply to SIL-SA bundles.

9.4.1 make_sil_sa_bundle

The **make_sil_sa_bundle** script should be used to create SIL-SA code stubs for multiple YANG modules. Just one SIL-SA library file will be produced for the entire bundle.



9.5 Static SIL-SA Libraries

The SIL-SA library for a module or a bundle can be built as a static library and linked directly into the “sil-sa-app” or “combo-app” programs (or an application based on these programs).

```
Quick Steps

1) make_sil_sa_dir or make_sil_sa_bundle
2) make STATIC=1
3) register static library in the SIL-SA application
```

There is no limit to the number of static SIL-SA libraries that can be linked into the program image or registered with the SIL-SA subsystem.

A static SIL-SA library is not used unless it is configured for use in the main server. The code is available at all times (since it is statically linked) and cannot be removed (only disabled).

Enable a static SIL-SA library

- Configuration parameter
  - --module=<module-name>
  - --bundle=<bundle-name>
- RPC operation
  - <load>
  - <load-bundle>

Disable a static SIL-SA library

- RPC operation
  - <unload>
  - <unload-bundle>

9.5.1 Static SIL-SA Library Example

This information is also in the README-STATIC_SILSA.txt file in the root of the source code directory.

```
IMPORTANT info on Using Static SIL-SA Libraries
================================================================================

0) Assume a SIL-SA library is setup
```

```
 > make_sil_sa_dir test2

This could also be a bundle (make_sil_sa_bundle)

1) Build a static version of the SIL-SA library

 > cd test2
 > make STATIC=1

2) Copy the static library to a common directory if desired.
   This is not required but may make using the STATIC_SILSA variable easier.

 > cp lib/libtest2.a $HOME/silsa/libtest2_sa.a

3) Use the macro STATIC_SILSA when building the sil-sa-app

Option 1: Change in Makefile:

  STATIC_SILSA=-L /home/andy/silsa -ltest2_sa

Option 2: set from command line:

  > STATIC_SILSA='-L /home/andy/silsa -ltest2_sa' make

Note that the -L part must be first and it must specify where the static
libraries are located.  One or more -l parts can follow.
The -l parameter does not use the full name. Instead libtest2_sa.a can
be specified as -ltest2_sa.

4) Register the static library in the sil-sa-app or combo-app program

Example from sil-sa-app/main.c:

/* extern definitions for the 3 expected callbacks */
AGT_SIL_LIB_EXTERN(test2)

static status_t static_silsa_init (void)
{
    /* example: module=test2;
     * need to use in Makefile (example)
     * STATIC_SILSA=-L /home/andy/silsa -ltest2_sa
     * The actual library names are not needed in this code
     */
    status_t res =
        agt_sil_lib_register_statlib((const xmlChar *)"test2",
                                     y_test2_init,
                                     y_test2_init2,
                                     y_test2_cleanup);

    return res;
}

5) The SIL-SA library will not be used unless the module or bundle is loaded.

   > load test2
   > netconfd-pro module=test2
```

9.6 SIL Makefile

The SIL-SA Makefile is based on the Makefile for YumaPro sources. The automake program is not used at this time. There is no ./configure script to run. There are 2 basic build steps:

1. **make [DEBUG=1] [STATIC=1]**
2. **[sudo] make install**

The installation step may require root access via the **sudo** program, depending on the system.

9.6.1 Target Platforms

The following target platforms are supported:

- **Fedora, Ubuntu, CentOS**: These are the default targets and no special command line options are needed.

9.6.2 Build Targets

The following table describes the build targets that are supported:

YumaPro Build Targets

Make Target	Description
all	Make everything. This is the default if not target is specified.
depend	Make the dependencies files.
clean	Remove the target files.
superclean	Remove all the dependencies and the target files.
distclean	Remove all the distribution files that make be installed, all the dependencies and the target files.
test	Make any test code.
lint	Run a lint program on the source files.
install	Install the components into the system.
uninstall	Remove the components from the system.

9.6.3 Command Line Build Options

There are several command line options that can be used when making the YumaPro source code. These may have an impact on building and linking the SIL source code. The top-level README file in the source code contains the latest definition of the make flags.

9.6.4 Example SIL Makefile

The script **/usr/bin/make_sil_dir_pro** or **/usr/bin/make_sil_bundle** is used to automatically create a SIL work sub-directory tree. Support files are located in the **/usr/share/yumapro/util** directory.

The Makefiles for SIL and SIL-SA modules are located in the “util” directory

- **Default SIL Makefile location:** /usr/share/yumapro/util

9.7 Controlling SIL Behavior

There are several extensions and API functions that control how the server will invoke the SIL code and process data in edit requests

9.7.1 SIL Invocation Order (sil-priority)

It is possible for the client to send several edits within a single <edit-config> or <copy-config> request.

Sometimes it is useful to make sure certain data structures are validated and processed before other data structures.

The sil-priority extension can be used to control the order that edits are processed. This extension is defined in section 12.2 “Built-in YANG Language Extensions”.

The SIL priority can also be set at run-time with an API function. This allows SIL priority to be setup in platform-specific ways, without needing to alter the YANG file for this purpose.

```
/********************************************************************
* FUNCTION agt_cb_set_sil_priority
*
* Set the desired SIL priority with a callback instead of
* using the YANG extension for this purpose
*
* INPUTS:
*   defpath == Xpath with default (or no) prefixes
*   sil_priority = SIL priority value to use (1..255)
* RETURNS:
*   status
********************************************************************/
extern status_t
    agt_cb_set_sil_priority (const xmlChar *defpath,
                             uint8 sil_priority);
```

Example: Set the SIL priority to 30 for the /interfaces/interface list.

```
status_t res =
  agt_cb_set_sil_priority((const xmlChar *)”/if:interfaces/if:interface”,
                          30);
```

9.7.2 Reverse SIL priority for delete operations (--sil-prio-reverse-for-deletes)

It is possible for the client to send several edits within a single <edit-config> request. Sometimes it is useful to make sure certain data structures are validated and processed before other data structures. The "sil-priority" extension can be used to control the order of the edits and how they are processed.

In a case where you set "sil-priority" for multiple specific objects and want the order to be reversed during the DELETE operations there is a netconfd-pro configuration parameter "sil-prio-reverse-for-deletes".

If set to "true" the "sil-prio-reverse-for-deletes" parameter for netconfd-pro dictates that the SIL priority for DELETE operations will be reversed. This parameter can be used to delete leafref nodes with referenced by node in reverse order. If 'false' then the SIL priority will not be reversed. The default is "false".

When the parameter is set to "true" the server will invoke callbacks for the specific objects with "sil-prority" configured in reverse order.

The "sil-prio-reverse-for-deletes" parameter has following facts that should be noted:

- It will NOT change the "sil-priority" of the specific objects, it will instead reverse the "sil-priority" and the server will invoke the callbacks in reverse order
- This parameter will reverse "sil-priority" only if the edit operation is DELETE or REMOVE
- The server will not reverse the "sil-priority" for EDIT2 MERGE mode with mixed edits (if there is delete and other operations). Since the real operation is in the children objects and the server invokes a single callback for all the children edits at once.

Please note that the EDIT2 MERGE edits will not change on the invocation order if the callbacks have mixed operations on children (if there are delete on one child and other edit operation on second child), they will not be reversed anyhow.

The EDIT2 MERGE mode combines multiple edits in one callback and the server invoke just one callback for all the edits. Hence, the server has no any control of when it should reverse the priority for this callback in case of mixed children operations. The priority in this case will remain the same and the server will warn a user with following warning:

```
SIL priority for object 'uint16-leaf' set to 100.100
Warning: Cannot reverse SIL priority for EDIT2 child 'test:uint16-leaf'
```

However, in case the EDIT2 MERGE mode has only delete or remove operations on children the priority for this callback will be reversed.

It is recommended to use EDIT1 callbacks to fully control the reverse priority behavior. Or as an alternative it is recommended to avoid the EDIT2 MERGE mode with mixed operations if you want to reverse the priority of the edits.

Also, the server will reverse the edits priorities in case the EDIT2 children edit is the edit to delete or remove the default node. And if there is no any other children edits with not equal to remove or delete, or merge edit operation in case of default node removal.

9.7.3 Deletion of Child Nodes (sil-delete-children-first)

If the client deletes a container of list entries, then the server will normally only invoke the SIL callback for deletion for the container. It may be easier to allow the server to use the SIL callbacks for the child list nodes, to delete a data structure "bottom-up".

Use the sil-delete-children-first extension to force the server to delete all the child nodes before deleting the parent container.

The sil-delete-children-first extension can be used in any parent node (container or list) and can be used to force the server to invoke the SIL callback for any child node (container, list, leaf, leaf-list, anyxml).

9.7.4 Suppress leafref Validation

Leafref validation involves searching all the “pointed at” leaf instances to make sure that the leafref leaf being validated contains one of the values actually in use. An instance is required to exist for the leafref to be valid.

YANG 1.0 does not support the “require-instance” sub-statement for a leafref definition. In order to force the server to skip this validation step, use the **agt_cb_skip_leafref_validation** API function.

```
/*********************************************************************
* FUNCTION agt_cb_skip_leafref_validation
*
* Set a previously registered callback as a node
* that the server should skip leafref validation
* in order to save processing time
*
* INPUTS:
*   defpath == Xpath with default (or no) prefixes
*
* RETURNS:
*   status
*********************************************************************/
extern status_t
    agt_cb_skip_leafref_validation (const xmlChar *defpath);
```

Specify the path for the leafref node that should have validation skipped.

In the following example, the path “**/t641-1:B/t641-1:BB/t641-1:fd-mode**” represents a leafref leaf that will be treated as if the “**require-instance false**” statement was present in the YANG definition:

```
Inside SIL init function:


    res = agt_cb_register_callback(
        y_test_fd641_1_M_test_fd641_1,
        (const xmlChar *)"/t641-1:B/t641-1:BB/t641-1:fd-mode",
        y_test_fd641_1_R_test_fd641_1,
        test_fd641_1_B_BB_fd_mode_edit);
    if (res != NO_ERR) {
        return res;
    }

    res = agt_cb_skip_leafref_validation(
        (const xmlChar *)"/t641-1:B/t641-1:BB/t641-1:fd-mode");
    if (res != NO_ERR) {
        return res;
    }
```

9.8 SIL-SA APIs

The SIL-SA code is different from SIL code and some callbacks cannot provide the same set of pointers to use during the invocation, for example Transaction Complete and Start Hook Callbacks cannot provide Transaction Control Block in SIL-SA version of the callback since the Transaction Control Block is not available in the SIL-SA subsystem. It is only part of the **netconfd-pro** server and only accessible from the SIL code.

However, to supply as much information to the SIL-SA callbacks and provide as much possible the same functionality as in the SIL callbacks there are multiple API that can be used to achieve SIL like functionality in the SIL-SA code:

Function	Description
sil_sa_get_username()	Get the user_id value from the message header
sil_sa_get_client_addr()	Get the client address (client_addr value from the message header)
sil_sa_get_rpc_msg_id()	Get the rpc message id (txid_str) from the message header
sil_sa_set_error_msg()	Set Custom Error Message string in case of error

9.8.1 Access MSG Values

The following API provides a way to access the User Identifier and Client Address that are stored in the message header in regular SIL code.

```
      /* Get the user_id value from the message header */
      const xmlChar *user = sil_sa_get_username();

      /* Get the client address (client_addr value from the message header) */
      const xmlChar *client_addr = sil_sa_get_client_addr();
```

These two APIs only available in the SIL-SA version of the SIL code and intended to provide an access to specific fields in the message header since the message header itself is not available in the SIL-SA version of of the SIL.

9.8.2 SIL-SA EDIT APIs

The following API provides a way to access the Message ID (txid_str). This API is only available to EDIT callbacks.

It will be NULL for all the other callbacks.

```
      /* Get the rpc message id (txid_str) from the rpc_sil_sa_cb */
      const xmlChar *msg_id =  sil_sa_get_rpc_msg_id(msg);
```

The following API provides a way to set a Custom Error message. In order to set a custom Error Message to be sent to the server the following API can be used. This API is only available to EDIT callbacks.

```
      /* Get the rpc message id (txid_str) from the rpc_sil_sa_cb */
      const xmlChar *client_addr =  sil_sa_get_rpc_msg_id(msg);
```

For example, assume we have the following simplified but functional YANG module and we are making a SIL-SA code for this module as follow:

```
module error-example {
  namespace "http://netconfcentral.org/ns/error-example";
  prefix "err";

  revision 2021-11-19 {
    description
      "Example module";
  }

  container test-silsa-errors {                      // EDIT2 CB
    presence "Presence";
```

```
    leaf test-error {
      type string;
    }
  }
}
```

Assume the SIL-SA code reports an error in case the **test-error** leaf creation and when the value of a new leaf is **force-error.**

In this case there is an API **sil_sa_set_error_msg()** to set up a custom error to be returned to the server.

```
status_t
    u_err_test_silsa_errors_edit (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    agt_cbtyp_t cbtyp,
    op_editop_t editop,
    val_value_t *newval,
    val_value_t *curval)
{
    val_value_t *child_val = NULL;
    if (newval) {
        child_val =
            val_find_child(newval,
                           (const xmlChar *)"error-example",
                           (const xmlChar *)"test-error");
    }

    /* Get the rpc message id (txid_str) from the rpc_sil_sa_cb */
    const xmlChar *msg_id = sil_sa_get_rpc_msg_id(msg);

    /* Get the user_id value from the message header */
    const xmlChar *user = sil_sa_get_username();

    /* Get the client address (client_addr value from the message header) */
    const xmlChar *client_addr = sil_sa_get_client_addr();

    if (LOGDEBUG3) {
        log_debug3("\nEnter u_err_test_silsa_errors_edit");
        log_debug3_append("\nmsg_id:%s", msg_id);
        log_debug3_append("\nuser:%s", user);
        log_debug3_append("\nclient_addr:%s", client_addr);
    }

    switch (cbtyp) {
    case AGT_CB_VALIDATE:
        /* description-stmt validation here */
        break;
    case AGT_CB_APPLY:
        /* database manipulation done here */
        break;
    case AGT_CB_COMMIT:
        /* device instrumentation done here */
        switch (editop) {
        case OP_EDITOP_LOAD:
            break;
        case OP_EDITOP_MERGE:
            break;
        case OP_EDITOP_REPLACE:
```

```
            break;
        case OP_EDITOP_CREATE:

            if (child_val &&
                !xml_strcmp(VAL_STR(child_val), (const xmlChar *)"force-error")) {

                /* TRIGGER ERROR */
                res = ERR_NCX_OPERATION_NOT_SUPPORTED;

                sil_sa_set_error_msg(msg,
                                     (const xmlChar *)"SOME CUSTOM ERROR MSG");
            }

            break;
        case OP_EDITOP_DELETE:
            break;
        default:
            /* USE SET_ERROR FOR PROGRAMMING BUGS ONLY */
            res = SET_ERROR(ERR_INTERNAL_VAL);
        }
        break;
    case AGT_CB_ROLLBACK:
        /* undo device instrumentation here */
        break;
    default:
        /* USE SET_ERROR FOR PROGRAMMING BUGS ONLY */
        res = SET_ERROR(ERR_INTERNAL_VAL);
    }
    return res;

} /* u_err_test_silsa_errors_edit */
```

Thus, as a result of the following operation the SIL-SA agent will produce an error with a custom error message:

```
{
 "ietf-restconf:errors": {
  "error": [
   {
    "error-type":"application",
    "error-tag":"operation-not-supported",
    "error-app-tag":"no-support",
    "error-path":"/err:test-silsa-errors",
    "error-message":"SOME CUSTOM ERROR MSG",
    "error-info": {
     "error-number":273
    }
   }
  ]
 }
}
```

9.9 Short Names for SIL and SIL-SA Code Generation

Starting in release 20.10-5 the short-names" parameter is available to control long or short name generation for SIL and SIL-SA code.

- **long name:** contains the module name and the complete path name of the object in the name. This generates a completely unique function or identifier name that cannot conflict within the module or with any other module. However, the names can be very long and difficult to read or use in C code.
- **short name**: the module prefix or --force-prefix parameter value is used instead of the module name. The object name is used instead of the path component. If the name is already used then a suffix such as "_1" will be added to the name so that it is unique. A short name is a match if the prefix part and the path part are the same strings. A short name is unique within the module or bundle, and should be globally unique if the module prefixes are unique. A short name can change if the code is regenerated and the module has changed so that duplicate names have been added. E.g., "name" might change to "name_1" and the old "name_1" might change to "name_2".

Example:

```
Path: leaf /interfaces/interface/admin-status

GET2 long function: u_ietf_interfaces_interfaces_interface_admin_status_get

GET2 short function: u_if_admin_status_get
```

9.9.1 CLI Parameters

There are two yangdump-sdk (or make_sil* script) CLI parameters that control the short name generation

- **short-names**: Selects short names or long names
- **force-prefix**: Override the module prefix in short name generation. Ignored if --short-names=false.

```
       leaf force-prefix {
         description
           "If the 'short-names' parameter is 'true' then this
            object can be used to force a specific prefix
            value instead of the module prefix, when generating
            instrumentation code.

            This object should only be used if the module prefix is
            known to cause a naming conflict with existing code.

            This object cannot be used if the 'sil-bundle' parameter
            is also used, and --short-names=true.";
         type yt:NcxName;
       }


       leaf short-names {
         description
```

```
           "If 'true', generate instrumentation code using short
            names whenever possible. The module prefix and the
            node name will be used. A numeric qualifier will be
            appended to the name if the short name would cause
            a duplicate symbol to be generated. The 'force-prefix'
            value will be use for the prefix if that parameter is
            present.

            If 'false', then generate instrumentation code using
            long names which encode the module name and the
            entire path to the object into the name.";
          type boolean;
          default true;
        }
```

9.9.2 Short Name Summary

If the --short-names parameter is set to “true” (the default value) then short names will be used.

The header for each C file will contain the Short Name Mapping used.

SIL Example:

```
> make_sil_dir_pro --split ietf-interfaces --sil-get2 --sil-edit2
```

Example Short Name Summary from u_ietf-interfaces.c

```
Short Name Mappings
  admin_status = /interfaces/interface/admin-status
  admin_status_1 = /interfaces-state/interface/admin-status
  description = /interfaces/interface/description
  discontinuity_time = /interfaces/interface/statistics/discontinuity-time
  discontinuity_time_1 = /interfaces-state/interface/statistics/discontinuity-time
  enabled = /interfaces/interface/enabled
  higher_layer_if = /interfaces/interface/higher-layer-if
  higher_layer_if_1 = /interfaces-state/interface/higher-layer-if
  if_index = /interfaces/interface/if-index
  if_index_1 = /interfaces-state/interface/if-index
  in_broadcast_pkts = /interfaces/interface/statistics/in-broadcast-pkts
  in_broadcast_pkts_1 = /interfaces-state/interface/statistics/in-broadcast-pkts
  in_discards = /interfaces/interface/statistics/in-discards
  in_discards_1 = /interfaces-state/interface/statistics/in-discards
  in_errors = /interfaces/interface/statistics/in-errors
  in_errors_1 = /interfaces-state/interface/statistics/in-errors
  in_multicast_pkts = /interfaces/interface/statistics/in-multicast-pkts
  in_multicast_pkts_1 = /interfaces-state/interface/statistics/in-multicast-pkts
  in_octets = /interfaces/interface/statistics/in-octets
  in_octets_1 = /interfaces-state/interface/statistics/in-octets
  in_unicast_pkts = /interfaces/interface/statistics/in-unicast-pkts
```

```
    in_unicast_pkts_1 = /interfaces-state/interface/statistics/in-unicast-pkts
    in_unknown_protos = /interfaces/interface/statistics/in-unknown-protos
    in_unknown_protos_1 = /interfaces-state/interface/statistics/in-unknown-protos
    interface = /interfaces/interface
    interface_1 = /interfaces-state/interface
    interfaces = /interfaces
    interfaces_state = /interfaces-state
    last_change = /interfaces/interface/last-change
    last_change_1 = /interfaces-state/interface/last-change
    link_up_down_trap_enable = /interfaces/interface/link-up-down-trap-enable
    lower_layer_if = /interfaces/interface/lower-layer-if
    lower_layer_if_1 = /interfaces-state/interface/lower-layer-if
    name = /interfaces/interface/name
    name_1 = /interfaces-state/interface/name
    oper_status = /interfaces/interface/oper-status
    oper_status_1 = /interfaces-state/interface/oper-status
    out_broadcast_pkts = /interfaces/interface/statistics/out-broadcast-pkts
    out_broadcast_pkts_1 = /interfaces-state/interface/statistics/out-broadcast-pkts
    out_discards = /interfaces/interface/statistics/out-discards
    out_discards_1 = /interfaces-state/interface/statistics/out-discards
    out_errors = /interfaces/interface/statistics/out-errors
    out_errors_1 = /interfaces-state/interface/statistics/out-errors
    out_multicast_pkts = /interfaces/interface/statistics/out-multicast-pkts
    out_multicast_pkts_1 = /interfaces-state/interface/statistics/out-multicast-pkts
    out_octets = /interfaces/interface/statistics/out-octets
    out_octets_1 = /interfaces-state/interface/statistics/out-octets
    out_unicast_pkts = /interfaces/interface/statistics/out-unicast-pkts
    out_unicast_pkts_1 = /interfaces-state/interface/statistics/out-unicast-pkts
    phys_address = /interfaces/interface/phys-address
    phys_address_1 = /interfaces-state/interface/phys-address
    speed = /interfaces/interface/speed
    speed_1 = /interfaces-state/interface/speed
    statistics = /interfaces/interface/statistics
    statistics_1 = /interfaces-state/interface/statistics
    type = /interfaces/interface/type
    type_1 = /interfaces-state/interface/type
```

9.9.3 Constants

The constants generated in a SIL or SIL-SA H file will be affected by the short-names parameter.

Examples

Module and Revision Constants:

```
Long name constant:

#define y_ietf_interfaces_M_ietf_interfaces (const xmlChar *)"ietf-interfaces"
#define y_ietf_interfaces_R_ietf_interfaces (const xmlChar *)"2018-02-20"

Short name constant:

#define y_if_M_if (const xmlChar *)"ietf-interfaces"
#define y_if_R_if (const xmlChar *)"2018-02-20"
```

Feature Constants

```
Long name constant:

#define u_ietf_interfaces_F_arbitrary_names 1

Short name constant:

#define u_if_F_arbitrary_names 1
```

Data Structure Typedef Constants:

```
Long name constant:

/* leaf-list /interfaces/interface/lower-layer-if */
typedef struct y_ietf_interfaces_T_interfaces_interface_lower_layer_if_ {
    dlq_hdr_t qhdr;
    xmlChar *v_lower_layer_if;
} y_ietf_interfaces_T_interfaces_interface_lower_layer_if;

Short name constant:

typedef struct y_if_T_lower_layer_if_ {
    dlq_hdr_t qhdr;
    xmlChar *v_lower_layer_if;
} y_if_T_lower_layer_if;
```

Object Name Constant:

```
Long name constant:

#define y_ietf_interfaces_N_admin_status (const xmlChar *)"admin-status"

Short name constant:

#define y_if_N_admin_status (const xmlChar *)"admin-status"
```

9.9.4 Function and Variable Names

The function and variable names generated in a SIL or SIL-SA H file will be affected by the short-names parameter.

GET2 Example:

```
Long name:

extern status_t u_ietf_interfaces_interfaces_interface_admin_status_get (
    getcb_get2_t *get2cb,
    const xmlChar *k_interfaces_interface_name);

Short name:

extern status_t u_if_admin_status_get (
    getcb_get2_t *get2cb,
    const xmlChar *k_name);
```

EDIT2 Example:

```
Long name:

extern status_t u_ietf_interfaces_interfaces_interface_edit (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    agt_cbtyp_t cbtyp,
    op_editop_t editop,
    val_value_t *newval,
    val_value_t *curval,
    const xmlChar *k_interfaces_interface_name);

Short name:

extern status_t u_if_interface_edit (
    ses_cb_t *scb,
    rpc_msg_t *msg,
    agt_cbtyp_t cbtyp,
    op_editop_t editop,
    val_value_t *newval,
    val_value_t *curval,
    const xmlChar *k_name);
```

10 YControl Subsystem

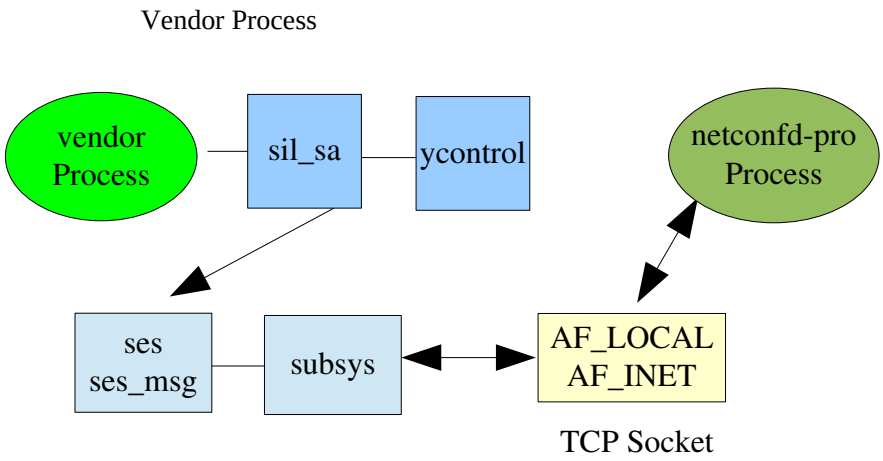
The YControl subsystem is used to allow asynchronous operation between **netconfd-pro** and higher layer services, such as **SIL-SA**. The YControl subsystem is a shared library that is linked with other libraries and runs in the vendor process.

The YControl subsystem is designed to support multiple independent upper-layer services . The main server does not provide an external API to the server handler for each subsystem at this time.

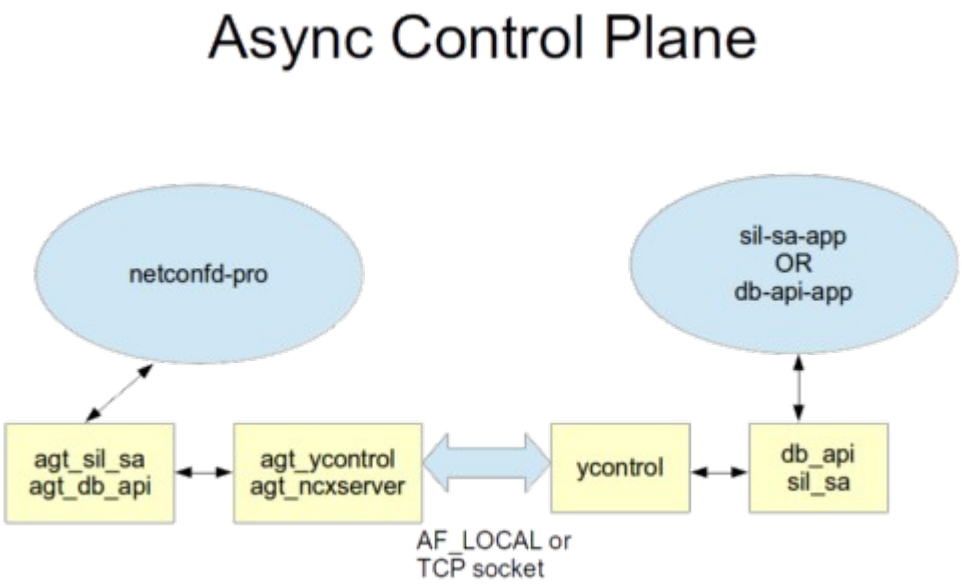
An IO poll API is called periodically by the vendor process to check for incoming messages.

The subsystem provides a communication service between the server and another process. It does not provide any high-level service within the server. It handles the following services:

- connection and reconnection to the **netconfd-pro** server
- configurable connect and retry parameters
- the server can boot before or after the process using **YControl**.
- registration API for higher layer services
- message processing API for higher layer services



Subsystem Software Components



Server to Subsystem Service Layer

10.1 YControl API Functions

Each high level service must register the proper callback functions with the YControl subsystem.

The application will poll the IO function periodically to handle communication with the main server. The callback functions registers with the subsystem will be invoked as needed, based on the received message.

The main YControl functions are described in this section.

10.1.1 Initialization and Cleanup

The following functions are used for initialization and cleanup of the YControl subsystem:

- **ycontrol_init:** Phase 1 Initialization
- **ycontrol_init2:** Phase 2 Initialization
- **ycontrol_register_service:** Register a set of callback functions fort a high-level service
- **ycontrol_cleanup:** Shutdown YControl subsystem and all upper layer services
- **ycontrol_request_shutdown:** Trigger shutdown of YControl subsystem and all upper layer services
- **ycontrol_shutdown_requested:** Check if shutdown of YControl subsystem has been triggered

```
/********************************************************************
* FUNCTION ycontrol_init
*
* Setup global vars before accepting any requests
*
* INPUTS:
*    argc == argument count
*     argv == argument array
*    subsys_id == sub-system identifier
* RETURNS:
*    status: 0 == OK
*********************************************************************/
extern status_t
    ycontrol_init (int argc,
                   char *argv[],
                   const xmlChar *subsys_id);


/********************************************************************
* FUNCTION ycontrol_init2
*
* Setup connection to server
*
* RETURNS:
*    status
*********************************************************************/
extern status_t
    ycontrol_init2 (void);



/********************************************************************
* FUNCTION ycontrol_register_service
```

```
*
* Register a specific service with the YumaPro control message manager
*
* INPUTS:
*
* RETURNS:
*    status
*********************************************************************/
extern status_t
    ycontrol_register_service (const xmlChar *service_name,
                               ycontrol_service_start_t service_start,
                               ycontrol_service_stop_t service_stop,
                               ycontrol_service_msg_rcvr_t service_rcvr,
                               ycontrol_service_shutdown_t service_shut);



/*********************************************************************
* FUNCTION ycontrol_cleanup
*
* Cleanup ycontrol layer
*
* INPUTS:
*   profile == service profile to cleanup
*
*********************************************************************/
extern void
    ycontrol_cleanup (void);


/*********************************************************************
* FUNCTION ycontrol_request_shutdown
*
* Request a control message handler shutdown
*
*********************************************************************/
extern void
    ycontrol_request_shutdown (void);


/*********************************************************************
* FUNCTION ycontrol_shutdown_requested
*
* Check if a control message handler shutdown is in progress
*
* RETURNS:
*    TRUE if shutdown mode has been started
*
*********************************************************************/
extern boolean
    ycontrol_shutdown_requested (void);
```

10.1.2 Runtime Functions

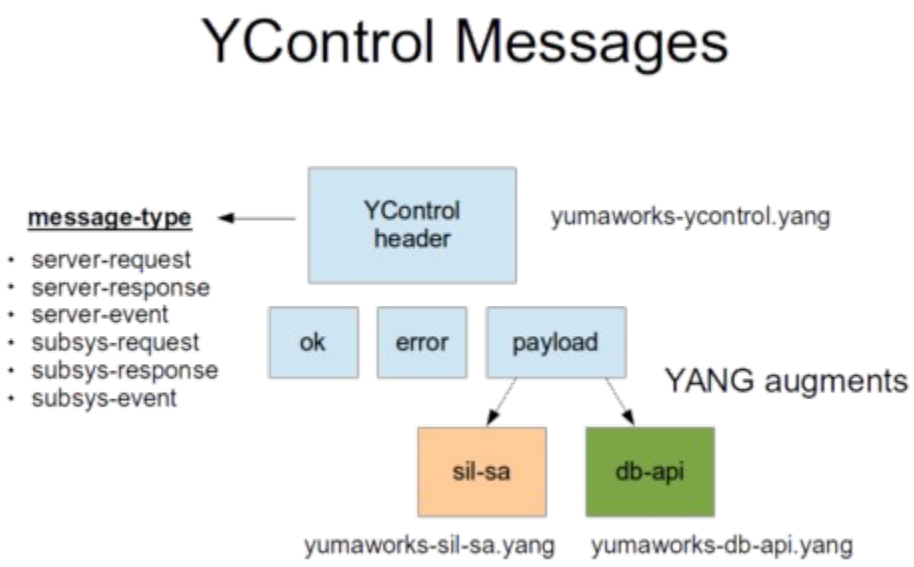
The following functions are used after initialization is complete to access the YControl subsystem:

- **ycontrol_check_io:** Check for any incoming messages from the main server
- **ycontrol_is_ready**: Make sure the YControl subsystem is ready for a new request
- **ycontrol_send:** Send any YControl message to the main server
- **ycontrol_send_error:** Send an error response to the main server

```
/********************************************************************
* FUNCTION ycontrol_check_io
*
* Check for input/output
*
* RETURNS:
*   status
********************************************************************/
extern status_t
    ycontrol_check_io (void);


/********************************************************************
* FUNCTION ycontrol_is_ready
*
* Check if the ycontrol ready is up and ready to be used
*
* RETURNS:
*   TRUE if ready; FALSE if not
********************************************************************/
extern boolean
    ycontrol_is_ready (void);



/********************************************************************
* FUNCTION ycontrol_send
*
* Send a YControl message
*
* INPUTS:
*   service_id == service sending this message
*   msgid == address of message ID;
*       for response this is non-zero
*   msgtype == type of YControl message to send
*   service_payload == val_value_t tree to add to message payload
*                  == NULL if not used
*    msg_status == NO_ERR or error status for message; ignored if
*        service_payload is non-NULL
* OUTPUTS:
*   if *msgid was zero on input:
*       *msgid set to the message ID assigned to the msg
* RETURNS:
*   status
********************************************************************/
extern status_t
    ycontrol_send (const xmlChar *service_id,
```

```
                   uint32 *msgid,
                   ycontrol_msgtype_t msgtype,
                   val_value_t *service_payload,
                   status_t msg_status);


/*****************************************************************
* FUNCTION ycontrol_send_error
*
* Send a YControl <error> message
*
* INPUTS:
*   service_id == service sending this message
*   msgid == address of message ID;
*       for response this is non-zero
*   msg_status == NO_ERR or error status for message; ignored if
*        service_payload is non-NULL
*   error_index == error index if sending an <error> response
*   error_msg == error message if sending an <error> response
* RETURNS:
*   status
*****************************************************************/
extern status_t
    ycontrol_send_error (const xmlChar *service_id,
                         uint32 *msgid,
                         status_t msg_status,
                         uint32 error_index,
                         const xmlChar *error_message);
```

10.2 YControl Message Structure



There are 6 types of messages supported, indicated by the message-type leaf

- **server-request**: request from server to subsystem, expecting a subsys-response
- **server-response**: response from server to subsystem to a subsys-request message
- **server-event**: event from server to subsystem; no response
- **subsys-request**: request from subsystem to server, expecting a server-response
- **subsys-response**: response from subsystem to server to a server-request message
- **subsys-event**: event from subsystem to server; no response

Each message type shares the same payload structure. There are 3 formats, specified by the **message-payload** choice-stmt in the YANG module.'

- **container payload**: message body
- **leaf ok**: ACK response
- **container error**: error response

10.2.1 yumaworks-ycontrol YANG Module

The **yumaworks-ycontrol** YANG module contains the **YControl** message structure.

```
module yumaworks-ycontrol {

    namespace "http://yumaworks.com/ns/yumaworks-ycontrol";

    prefix "yctl";

    import yuma-ncx { prefix ncx; }
    import yuma-types { prefix yt; }

    organization "YumaWorks, Inc.";

    contact
        "Support <support at yumaworks.com>";

    description
       "YumaPro control system message definition.";

    revision 2014-11-19 {
        description
          "Support '*' as the service-id to indicate a server
           event message that is intended for the YControl layer
           itself, called ALL_SERVICES.
           Add shutdown-event message for ALL_SERVICES";
    }

    revision 2014-04-08 {
        description
          "Initial version.";
    }

    container ycontrol {
      ncx:abstract;
      ncx:hidden;

      leaf message-id {
        mandatory true;
        type uint32 {
          range "1 .. max";
        }
        description
          "Message identifier.
           For server-response and subsys-response message types,
           this value is the same as the corresponding request
           message.";
      }

      leaf message-type {
        mandatory true;
        type enumeration {
          enum server-event {
            description
              "Message from server to sub-system.
               No response expected.";
          }
          enum server-request {
```

```
          description
            "Request message from server to sub-system.
             A response is expected.";
        }
        enum server-response {
          description
            "Response message from server to sub-system.
             Sent when subsys-req received.
             No response is expected";
        }
        enum subsys-event {
          description
            "Message from sub-system to server.
             No response expected.";
        }
        enum subsys-request {
          description
            "Request message from sub-system to server.
             A response is expected.";
        }
        enum subsys-response {
          description
            "Response message from sub-system to server.
             Sent when server-req received.
             No response is expected";
        }
        enum ycontrol-error {
          description
            "Response message from either sub-system or server.
             Sent when a recoverable YControl or service layer
             error occurs.

             If non-recoverable error, then session is dropped
             and no response is sent. Example error: message
             is for a service-id that does not exist.
             No response is expected";
        }

      }
      description "Message type";
    }

    leaf server-id {
      mandatory true;
      type union {
        type yt:NcxName;
        type string { length 0; }
      }
      description "Server identifier or empty if not known by subsys";
    }

    leaf subsys-id {
      mandatory true;
      type yt:NcxName;
      description "Subsystem identifier";
    }

    leaf service-id {
      mandatory true;
      type union {
        type yt:NcxName;
        type string {
          pattern '\*';
```

```
        }
      }
      description
        "Service identifier. The value '*' indicates a
         server-event message to all services. These messages
         are handled by the ycontrol library, not the individual
         service libraries.";
    }

    choice message-payload {
      mandatory true;
      container payload {
        description
          "This <payload> container is augmented with a
           service-specific container from other modules.";

        leaf shutdown-event {
          type empty;
          description
           "Message type: server-event;
            Purpose: The server is shutting down. Sent to
            all services (service-id = '*')

            Expected Response Message: none";
        }  // leaf shutdown-event
      }
      leaf ok {
        type empty;
        description
          "Sent when a request message is processed
           successfully and no data is needed in the
           response.";
      }
      container error {
        leaf error-number {
          mandatory true;
          type uint32;
          description "Internal error number";
        }

        leaf transaction-id {
          type string;
            description
              "Server specific transaction identifier.
               Sent from subsystem to server in
               subsys-response.
               It identifies the transaction in case multiple
               transactions are in progress at once.";
        }

        leaf error-message {
          type string;
          description
            "Internal error message, if different from
             get_error_string(error-number).";
        }

        leaf error-index {
          type uint32 {
            range "1 .. max";
          }
          description
            "Internal edit index number from <start-transaction>
```

```
                message. Set only if an edit-specific error occurred.";
          }
        }
      }  // choice message-payload
    }  // container ycontrol
}
```

11 SIL-SA Subsystem

The SIL-SA subsystem provides sub-agent support for SIL callbacks. It uses messages between the server and subsystem to distribute SIL operations across multiple remote subsystems.

To use the SIL-SA subsystem, initialize and run the sil-sa library API functions. See the **sil-sa-app** source file "**main.c**" for an example of the API calls needed to enable the SIL-SA subsystem.

If SIL-SA libraries are available to the subsystem, and the YANG modules are available to the main server and the submodule, then the SIL-SA callback registration and message handling will be automatically handled by the subsystem sil_sa and ycontrol libraries.

There is no mechanism at this time to configure separate YANG modules on each subsystem. Each subsystem is required to ignore SIL-SA service requests for objects it does not support. It is possible for multiple subsystems and the main module to register for the same object. Each callback must decide if it should handle the specific instance being requested (E.g. interface "eth0" or "eth3"). Subsystems need to ignore requests for unsupported instances by returning an "ok" message to the main server for transaction requests.

In the registration phase, the main server will send the complete module and bundle parameter list to each subsystem. Each subsystem will load its SIL-SA library code as required for that subsystem.

It is up to the developer to make sure each subsystem registers with the main server with a unique subsys-id string. Only one subsystem can use the default (subsys1), The **--subsys-id** CLI parameter can be used to set the subsytem identifier.

11.1 Example SIL-SA Application

```
*********************************************************************
* FUNCTION main
*
* This is an example SIL-SA service main function.
*
* RETURNS:
*   0 if NO_ERR
*   status code if error connecting or logging into ncxserver
*********************************************************************/
int main (int argc, char **argv)
{
#ifdef MEMORY_DEBUG
    mtrace();
#endif

    char *subsys = NULL;
    char *address = NULL;
    uint16 port = 0;
    uint16 retry_limit = 0;
    boolean if_notif=false;  // hidden parameter
    boolean ycontrol_done = FALSE;

    /* need to check for the subsys-id parm before
     * the system is initialized
     */
    status_t res = get_subsys_parm(argv, &subsys);
    if (res != NO_ERR) {
        print_usage();
    }
```

```
    /* 1) setup yumapro messaging service profile */
    if (res == NO_ERR) {
        if (subsys == NULL) {
            res = ycontrol_init(argc, argv,
                                (const xmlChar *)"subsys1");
        } else {
            res = ycontrol_init(argc, argv,
                                (const xmlChar *)subsys);
        }
        ycontrol_done = TRUE;
    }

    /* 2) register services with the control layer */
    if (res == NO_ERR) {
        res = sil_sa_register_service();
    }

#ifdef SIL_SA_APP_STATLIB_TEST
    /* 2B) setup any static SIL-SA libraries */
    if (res == NO_ERR) {
        res = static_silsa_init();
    }
#endif  // SIL_SA_APP_STATLIB_TEST


    /* get the CLI parameters after the system is initialized
     * so library parameter handled correctly
     */
    if (res == NO_ERR) {
        res =  get_cli_parms(argv, &address, &port, &if_notif, &retry_limit);
        if (res != NO_ERR) {
            print_usage();
        }
    }

    /* set the retry limit if provided */
    if ((res == NO_ERR) && (retry_limit > 0)) {
        ycontrol_set_retry_limit(retry_limit);
    }

    /* It is also possible to set the retry_interval but there is
     * no CLI parameter provided for this purposes
     * if (res == NO_ERR) {
     *     ycontrol_set_retry_interval(retry_int_milliseconds);
     * }
     */


    /* 3) do 2nd stage init of the control manager (connect to server) */
    if (res == NO_ERR) {
        if (address) {
            if (port == 0) {
                port = 2023;
            }
            res = ycontrol_init2_ha("server1", address, port);
        } else {
            res = ycontrol_init2();
        }
    }

    useconds_t usleep_val = 10000;  // 10K micro-sec == 1/100 sec
    boolean done = FALSE;
```

```
    /* 4) call ycontrol_check_io periodically from the main program
     * control loop
     */
#ifdef SIL_SA_APP_DEBUG
    int id = 0;
#endif  // SIL_SA_APP_DEBUG

#ifdef SIL_SA_APP_NOTIF_TEST
    uint32 loop_cnt = 0;
#endif // SIL_SA_APP_NOTIF_TEST

#ifdef SIL_SA_APP_TERM_MSG_TEST
    uint32 term_msg_loop_cnt = 0;
#endif // SIL_SA_APP_TERM_MSG_TEST

    while (!done && res == NO_ERR) {
#ifdef SIL_SA_APP_DEBUG
        if (LOGDEBUG3) {
            log_debug3("\nsil-sa-app: checking ycontrol IO %d", id++);
        }
#endif  // SIL_SA_APP_DEBUG

        res = ycontrol_check_io();

        if (ycontrol_shutdown_now()) {
            // YControl has received a <shutdown-event>
            // from the server subsystem is no longer active
            // could ignore or shut down YControl IO loop
            done = TRUE;
        }

        // Using sleep to represent other program work; remove for real
        if (!done && res == NO_ERR) {
            (void)usleep(usleep_val);
        }

#ifdef SIL_SA_APP_NOTIF_TEST
        loop_cnt++;
        if (loop_cnt % 50 == 0 && if_notif) {
            sil_sa_notif_test(10, 20, (const xmlChar *)"this is a test");
        }
#endif // SIL_SA_APP_NOTIF_TEST

#ifdef SIL_SA_APP_TERM_MSG_TEST
        term_msg_loop_cnt++;
        if (term_msg_loop_cnt % 100 == 0) {
            sil_sa_term_msg_test(term_msg_loop_cnt);
        }
#endif  // SIL_SA_APP_TERM_MSG_TEST

    }

    /* 5) cleanup the control layer before exit */
    if (ycontrol_done) {
        ycontrol_cleanup();
    }

#ifdef MEMORY_DEBUG
    muntrace();
#endif

    return (int)res;
```

```
}  /* main */
```

11.2 SIL-SA Message Format

The SIL-SA service uses several messages to interact with the **netconfd-pro** server.

These messages are defined in the **yumaworks-sil-sa** YANG module. The SIL-SA payload is defined as a YANG container that augments the YControl “message-payload” container.

11.2.1 SIL-SA Registration Message Flow

The SIL-SA service needs to initialize with the **netconfd-pro** server.

1. **SIL-SA to Server:** *config-request:*
   The SIL-SA service registers itself with the server and requests its configuration in 1 message.
2. **Server to SIL-SA:** *config-response:*
   The server responds to the config request with a list of modules and SIL-SA bundles that need to be loaded
3. **SIL-SA to Server:** *register-request:*
   The SIL-SA service registers the SIL-SA callbacks supported by the subsystem.
4. **Server to SIL-SA:** *ok:*
   The server responds to the config request with an <ok> or an <error> message

11.2.2 Server Edit Transaction Message Flow

The **netconfd-pro** server will initiate edit datastore transactions when the subsystem starts or restarts (load-config), or when clients edit the datastore via a network management protocol.

1. **Server to SIL-SA:** start-transaction*:*
   The server starts the transaction with the "validate" SIL-SA callback phase.
2. **SIL-SA to Server:** ok*:*
   The SIL-SA service responds to the start-transaction request with an <ok> or an <error> message
3. **Server to SIL-SA:** continue-transaction:
   The server continues the transaction with the "apply" SIL-SA callback phase.
4. **SIL-SA to Server:** ok*:*
   The SIL-SA service responds to the continue-transaction request with an <ok> or an <error> message
5. **Server to SIL-SA:** continue-transaction:
   The server continues the transaction with the "commit" or "rollback" SIL-SA callback phase.
6. **SIL-SA to Server:** ok*:*
   The SIL-SA service responds to the continue-transaction request with an <ok> or an <error> message. The SIL-SA service removes any saved state for this transaction.

Exception handling:

validate:
The server may send a "start-transaction" message with the "validate" flag set. The SIL-SA service will know that the transaction is only 1 message and not to expect a "continue-transaction" message from the server.

reverse-edit:
The server may send a "start-transaction" message with the "reverse-edit" flag set. The SIL-SA service will handle all 3 callback phases for the edit at once. Only 1 response message is sent to the server. This mode is used when a different subsystem responded with an error after the subsystem performing the reverse edit has already successfully completed the transaction. A new transaction is sent to the subsystem in this case to undo the edit.

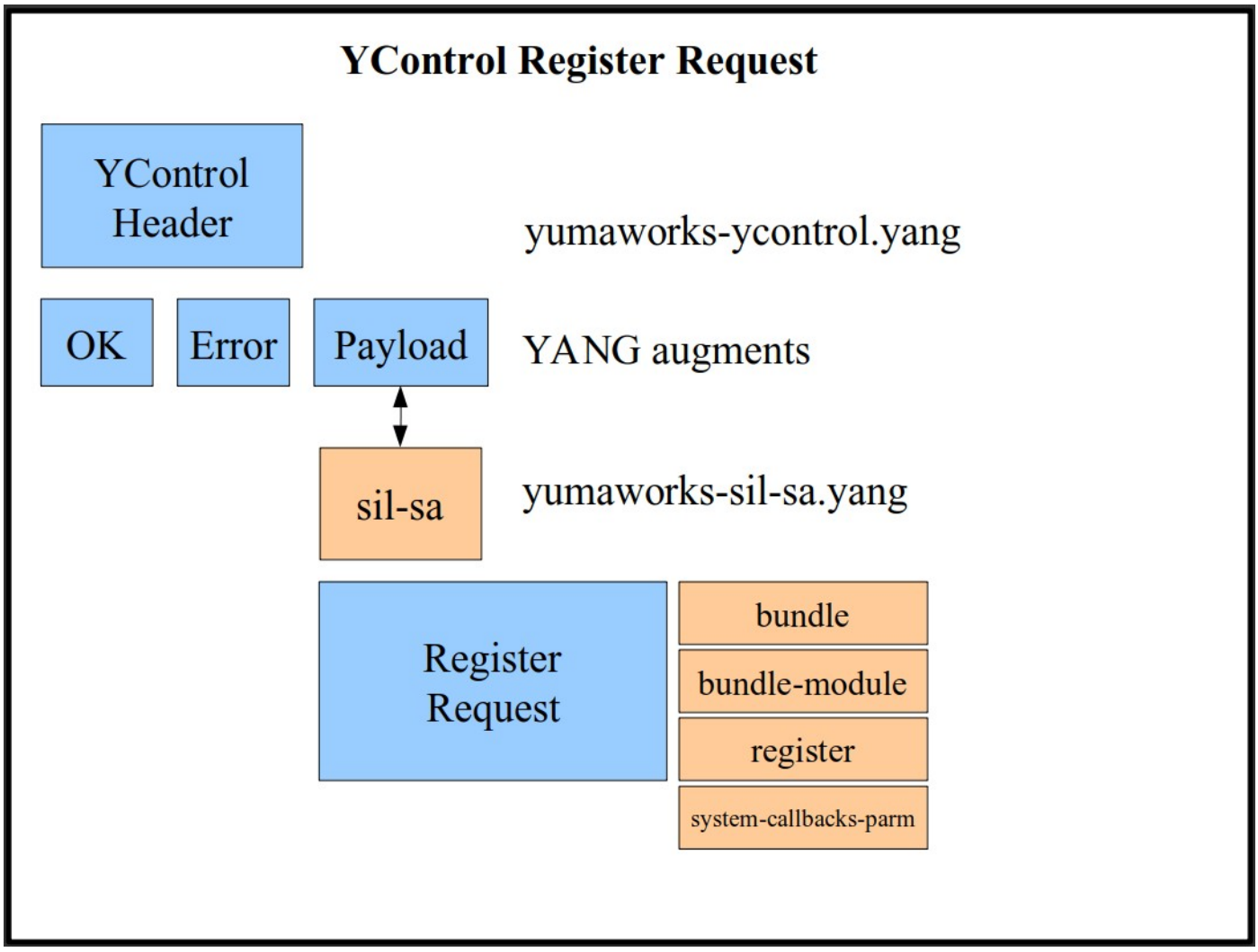
cancel-transaction:
The server may send a "cancel-transaction" message after a transaction has started. This can occur if another SIL-SA subsystem or main server SIL callback returned an error. Any error causes the transaction to be canceled. The SIL-SA service will not expect a "continue-transaction" message from the server.

11.2.3 SIL-SA Register Request Message

The **<register-request>** is a subsytem request message. The purpose of the message is to register the SIL-SA callback functions for this sub-system.

Expected Response Message: ok or error

The following diagram illustrates the **<register-request>** event message:



11.2.4 SIL-SA Start Transaction Message

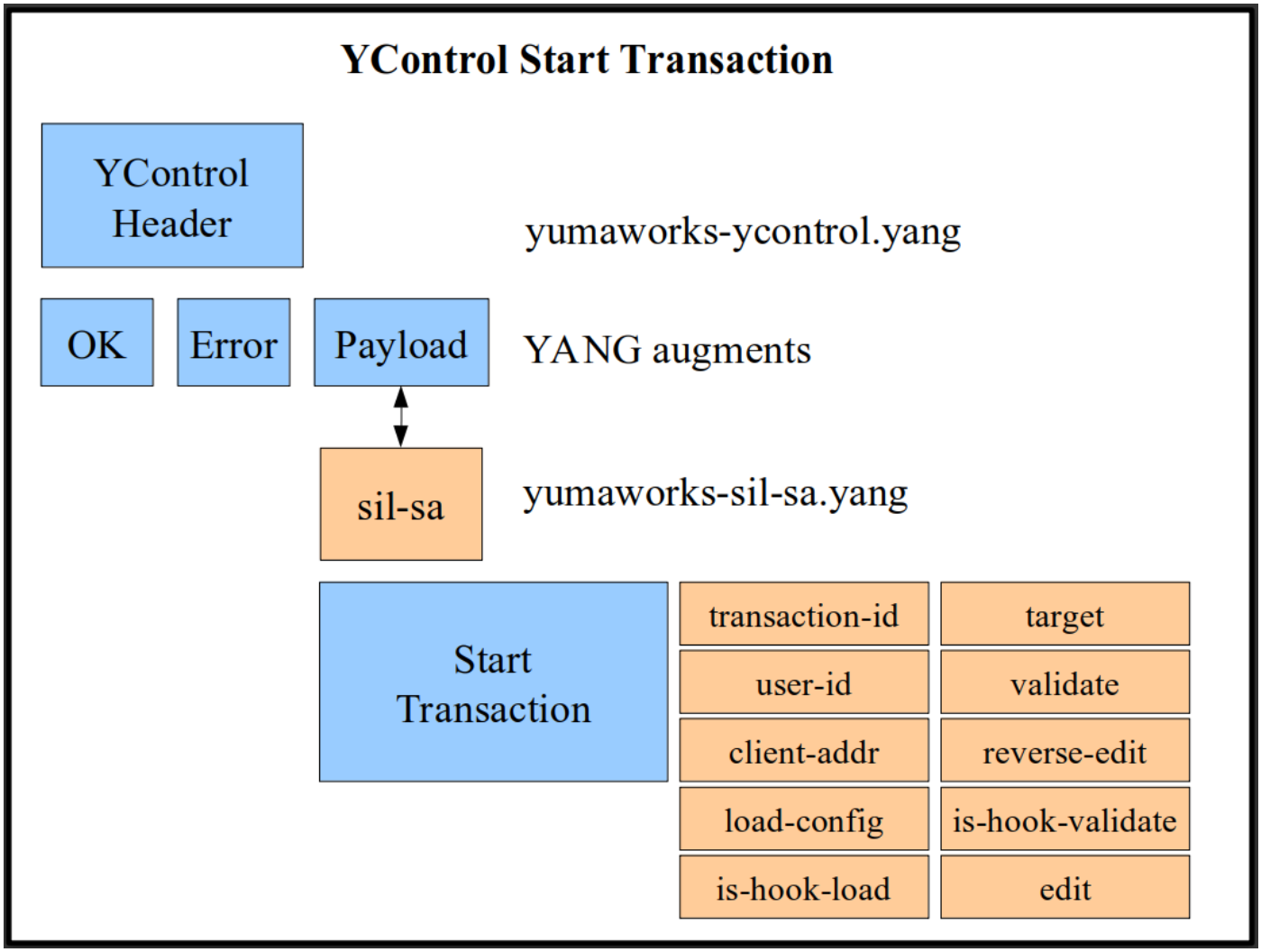
The **<start-transaction>** is a server request message. The purpose of the message is to start an edit transaction which may require the SIL-SA callback functions on the subsystem to be invoked.

This message requests that a new edit transaction be started on the subsystem. Only 1 transaction can be in progress at a time.

If this transaction is for a validate operation then there will not be any followup messages. Otherwise, the subsystem will retain this message until a cancel-transaction message has been received with the same transaction-id value, or a continue-transaction message has been received with the same transaction-id value for the 'rollback' or 'commit' phase.

Expected Response Message: transaction-response or error

The following diagram illustrates the **<start-transaction>** event message:

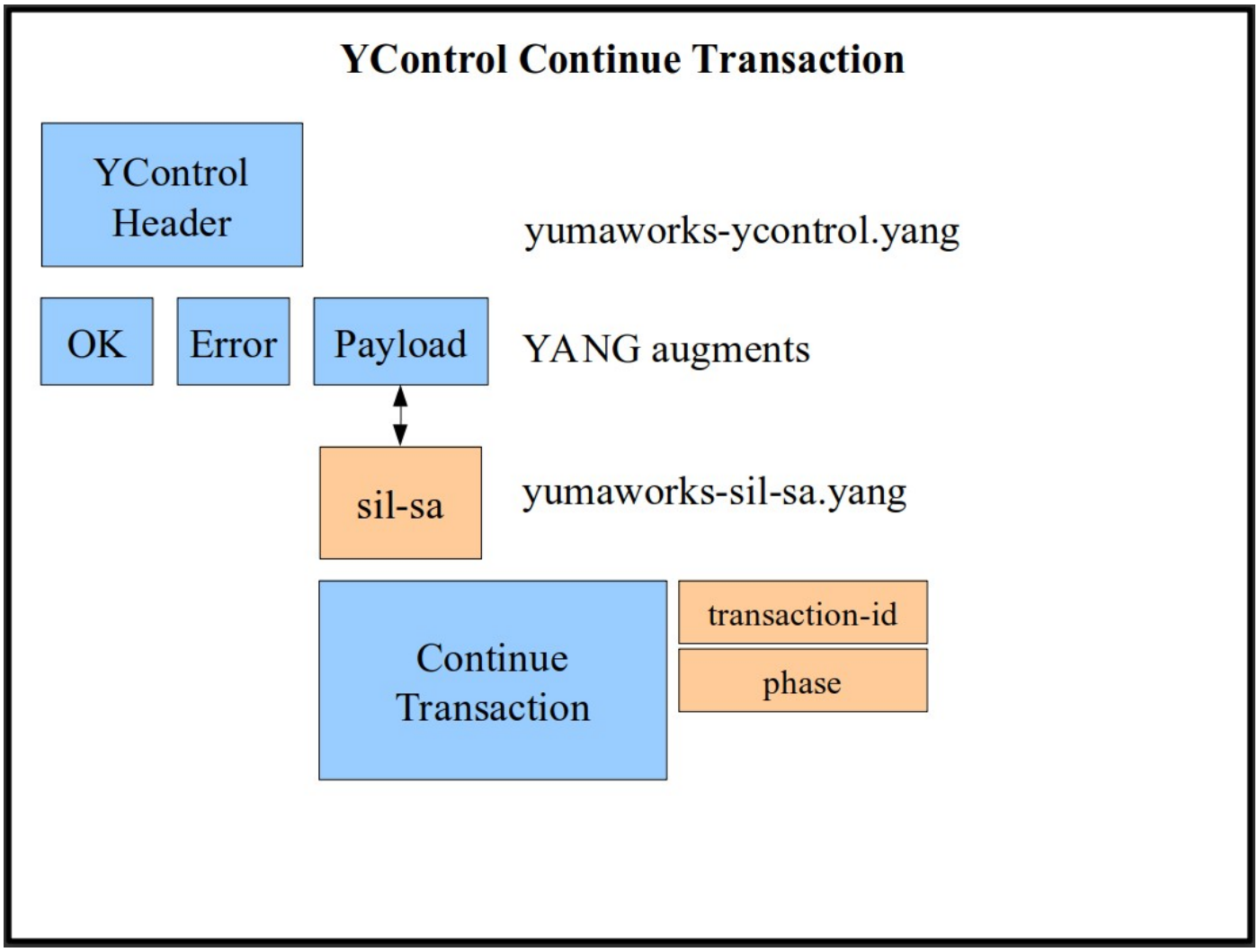


11.2.5 SIL-SA Continue Transaction Message

The **<continue-transaction>** is a server request message. The purpose of the message is to invoke a callback phase for an edit transaction in progress.

Expected Response Message: transaction-response or error

The following diagram illustrates the **<continue-transaction>** event message:

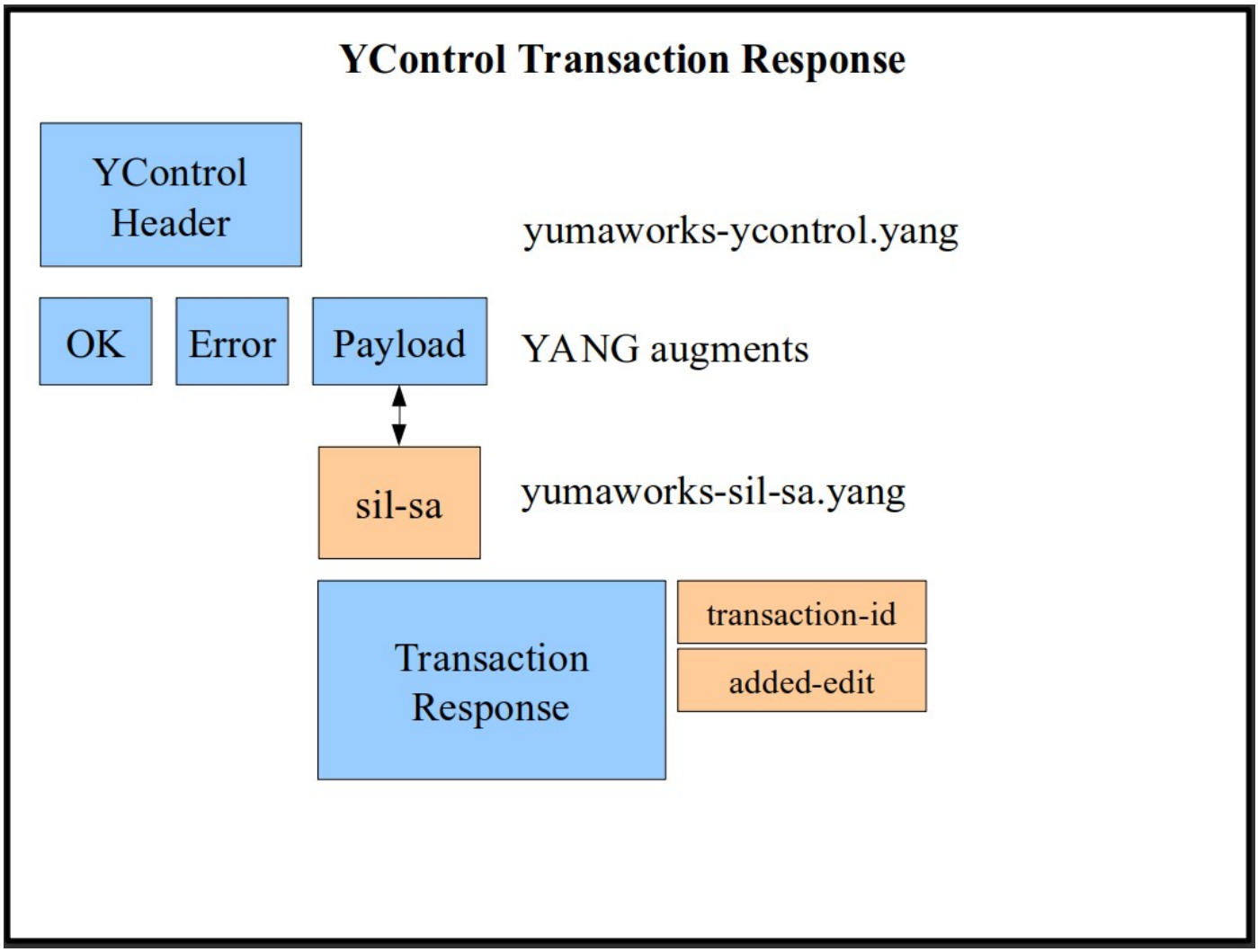


11.2.6 SIL-SA Transaction Response Message

The **<transaction-response>** is a subsystem response message. The purpose of the message is:

- Set Hook: return added edits data or status
- Post Set Hook: return added edits data or status
- Transaction Hook: Expected Response Message: ok or error
- If no Hook invoked: Expected Response Message: ok or error";

The following diagram illustrates the **<transaction-response>** event message:

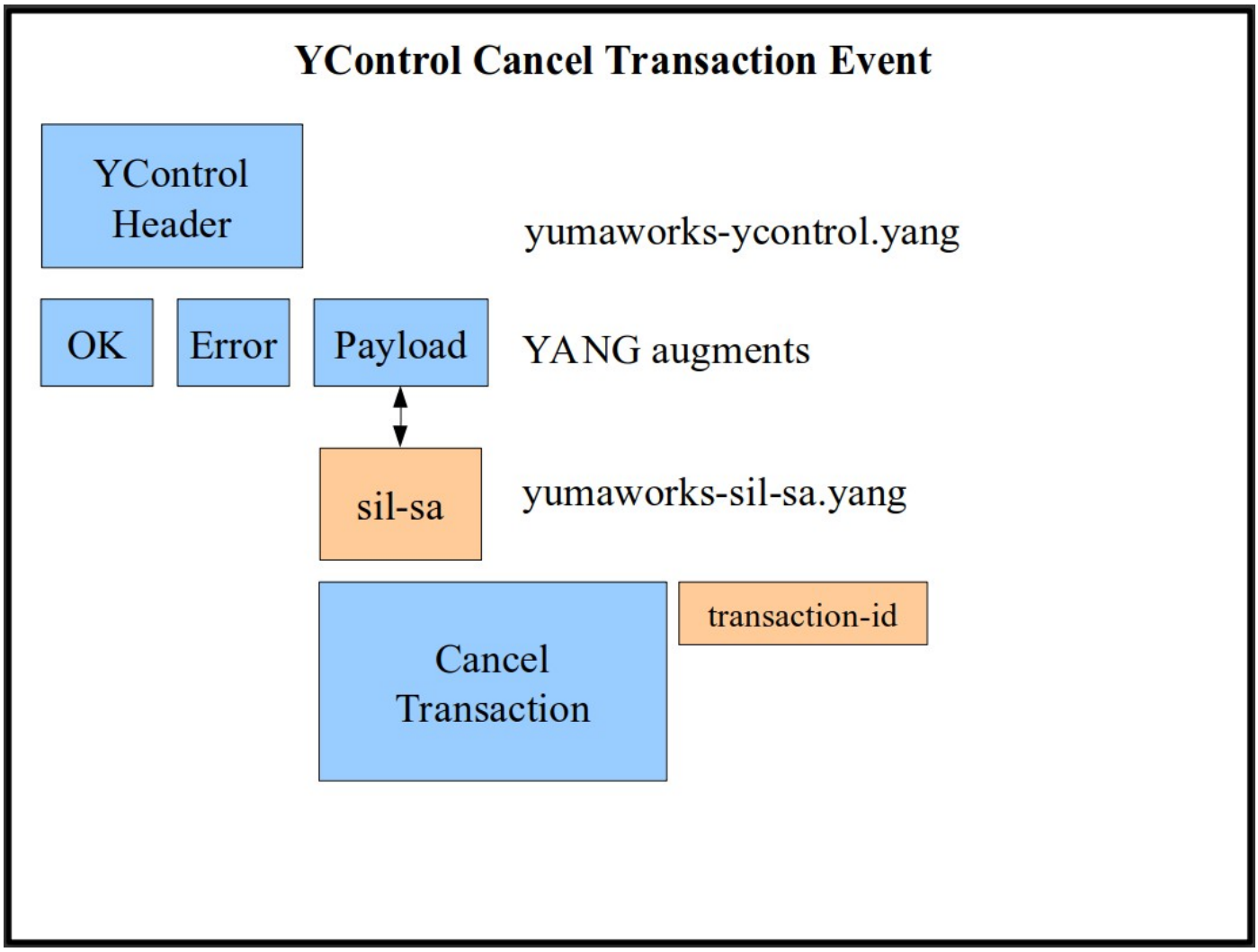


11.2.7 SIL-SA Cancel Transaction Event

The **<cancel-transaction>** is a server event message. The purpose of the message is to Cancel an edit transaction in progress.

Expected Response Message: none

The following diagram illustrates the **<cancel-transaction>** event message:

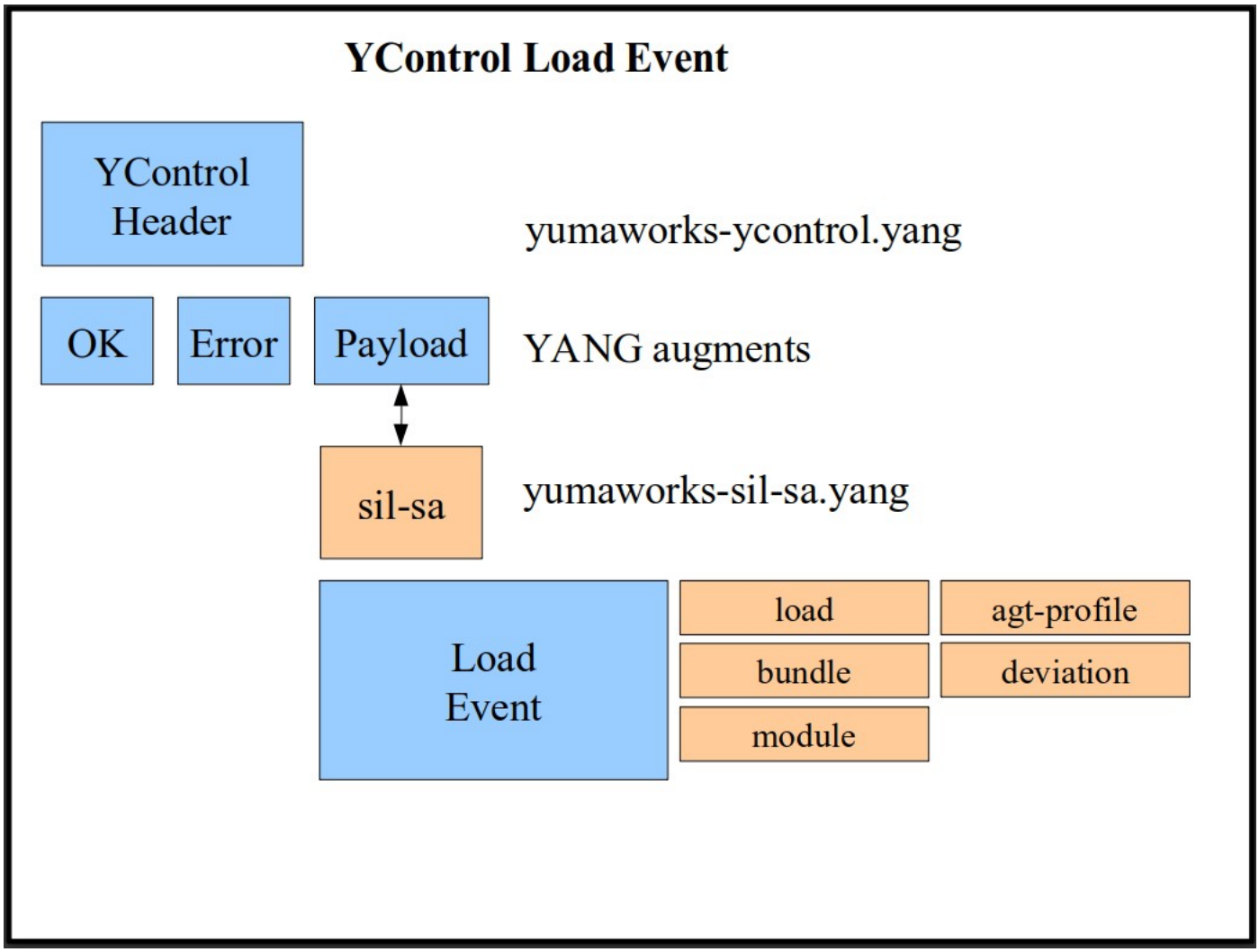


11.2.8 SIL-SA Load Event

The **<load-event>** is a server event message. The purpose of the message is to notify the SIL-SA subsytem that a module or bundle has been loaded or unloaded at run-time. Subsystem will load SIL-SA code and trigger a register event for any SIL calls registered.

Expected Response Message: none

The following diagram illustrates the **<load-event>** event message:

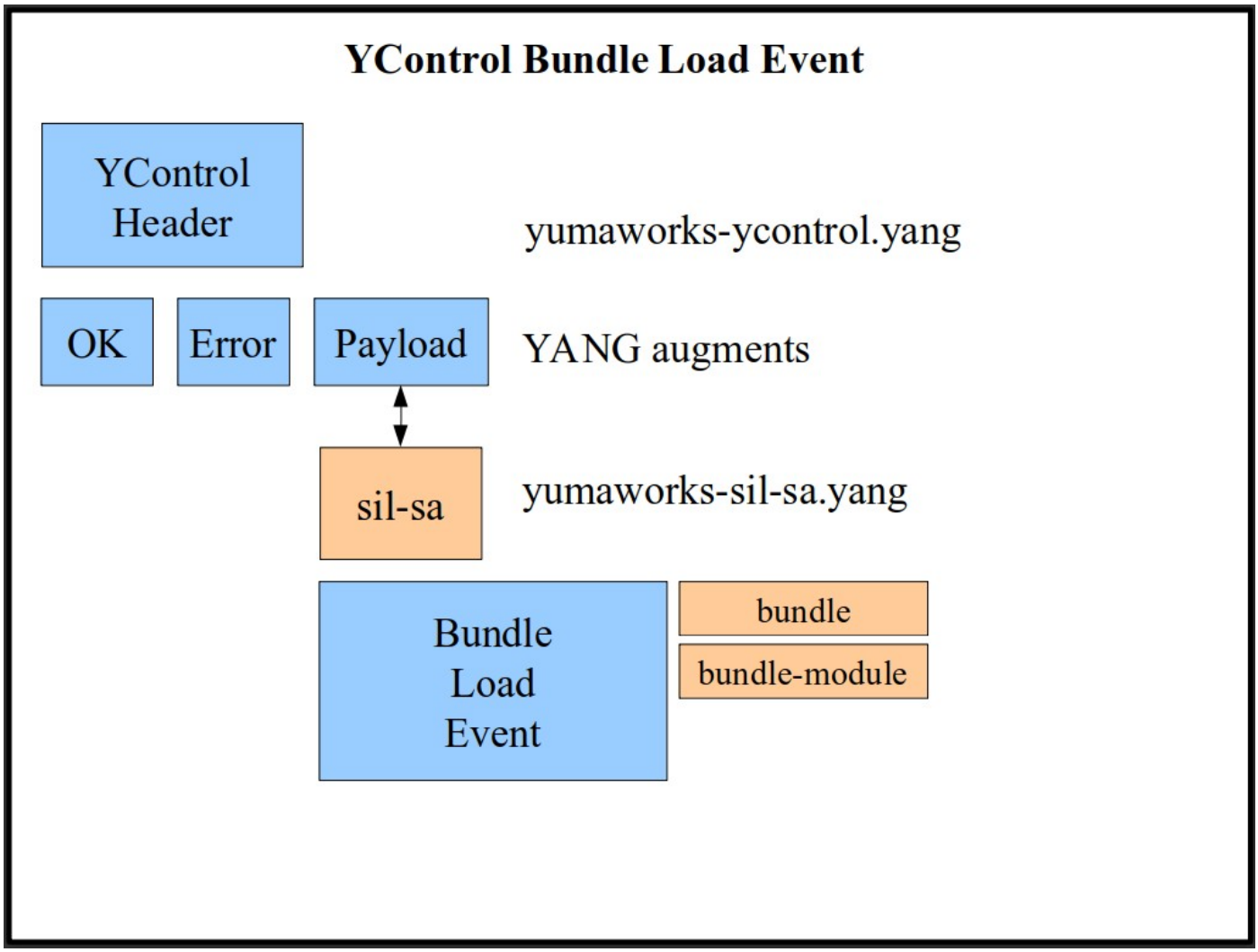


11.2.9 SIL-SA Bundle Load Event

The **<bundle-load-event>** is a subsytem event message. The purpose of the message is to notify the server that a SIL-SA bundle has been loaded with a load-event sent from the server. This has caused some modules to be loaded on the subsystem, that need to be reported back to the main server so the datastore validation, **agt_state**, and other system book-keeping can be done.

Expected Response Message: none

The following diagram illustrates the **<bundle-load-event>** event message:

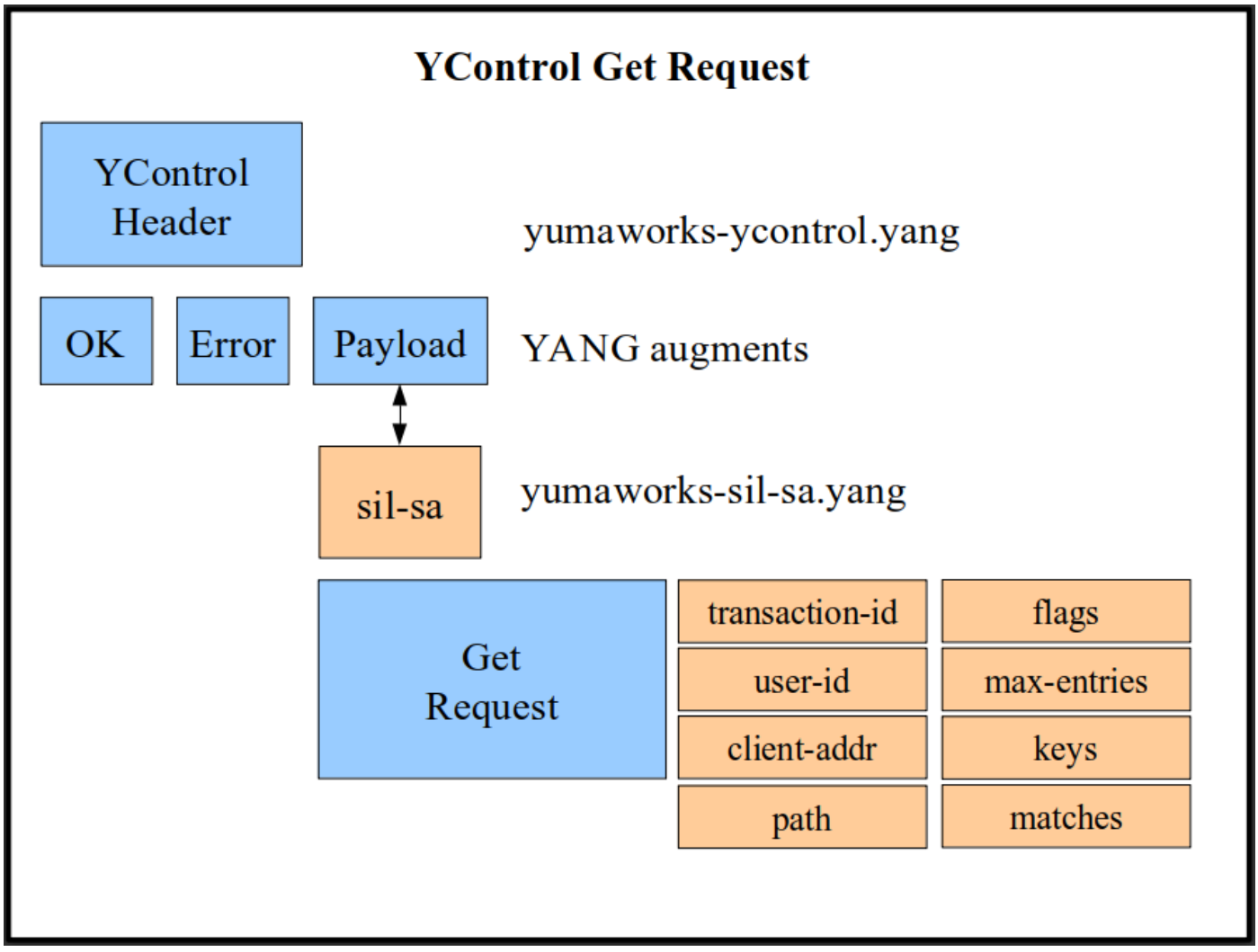


11.2.10 SIL-SA Get Request Message

The **<get-request>** is a server request message. The purpose of the message is to send a composite retrieval request to support NETCONF and RESTCONF get operations.

Expected Response Message: get-response or error

The following diagram illustrates the **<get-request>** event message:

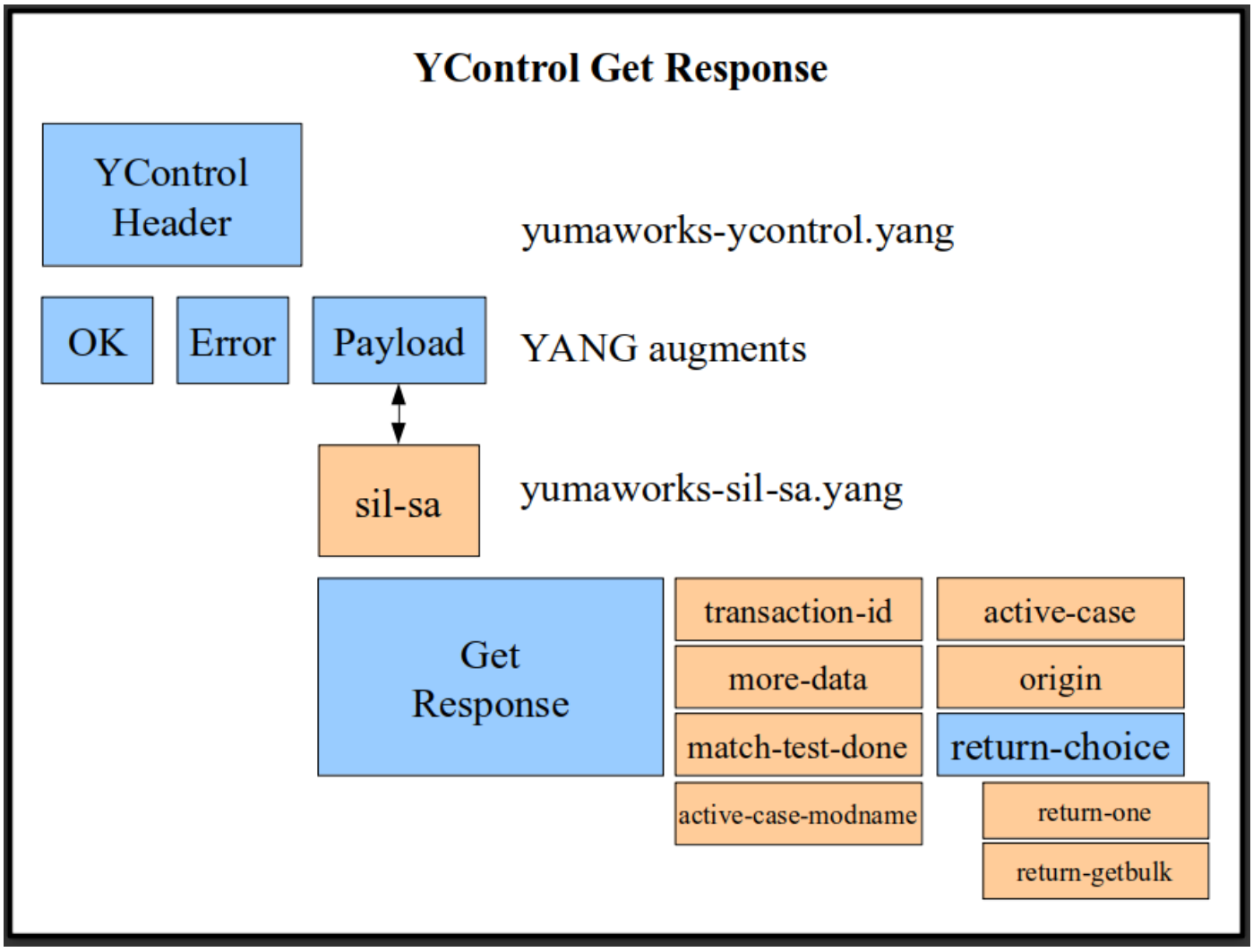


11.2.11 SIL-SA Get Response Message

The **<get-response>** is a subsystem response message. The purpose of the message is to send a subsystem generated composite retrieval response to support NETCONF and RESTCONF get operations.

Expected Response Message: none

The following diagram illustrates the **<get-response>** event message:

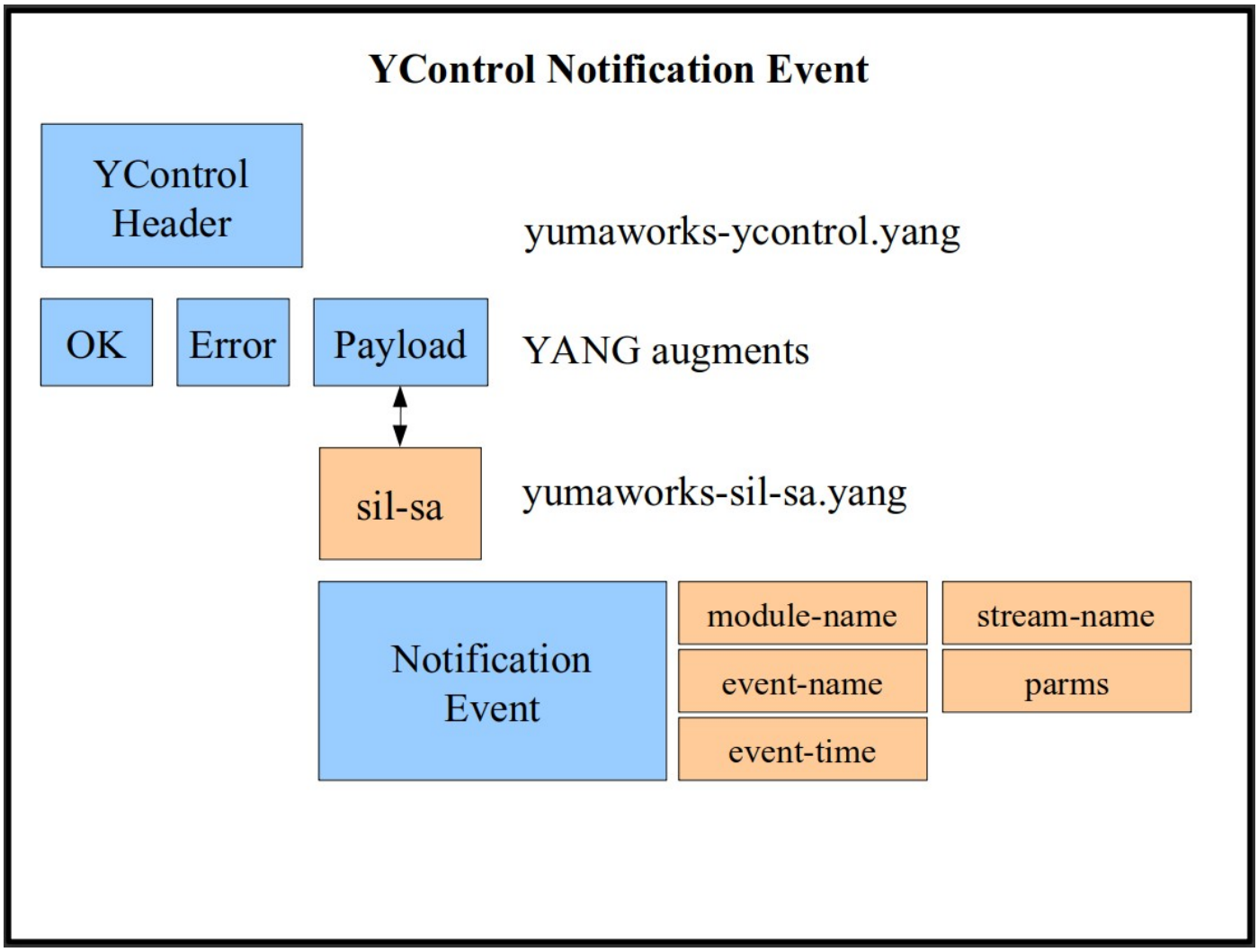


11.2.12 SIL-SA Notification Message

The **<notification>** is a subsystem event message. The purpose of the message is to send a subsystem generated YANG notification event for NETCONF and RESTCONF streams.

Expected Response Message: none

The following diagram illustrates the **<notification>** event message:



11.2.13 SIL-SA RPC Request Message

The **<rpc-request>** is a server request message. The purpose of the message is to start a RPC transaction which may require the require the SIL-SA callback functions on the subsystem to be invoked.

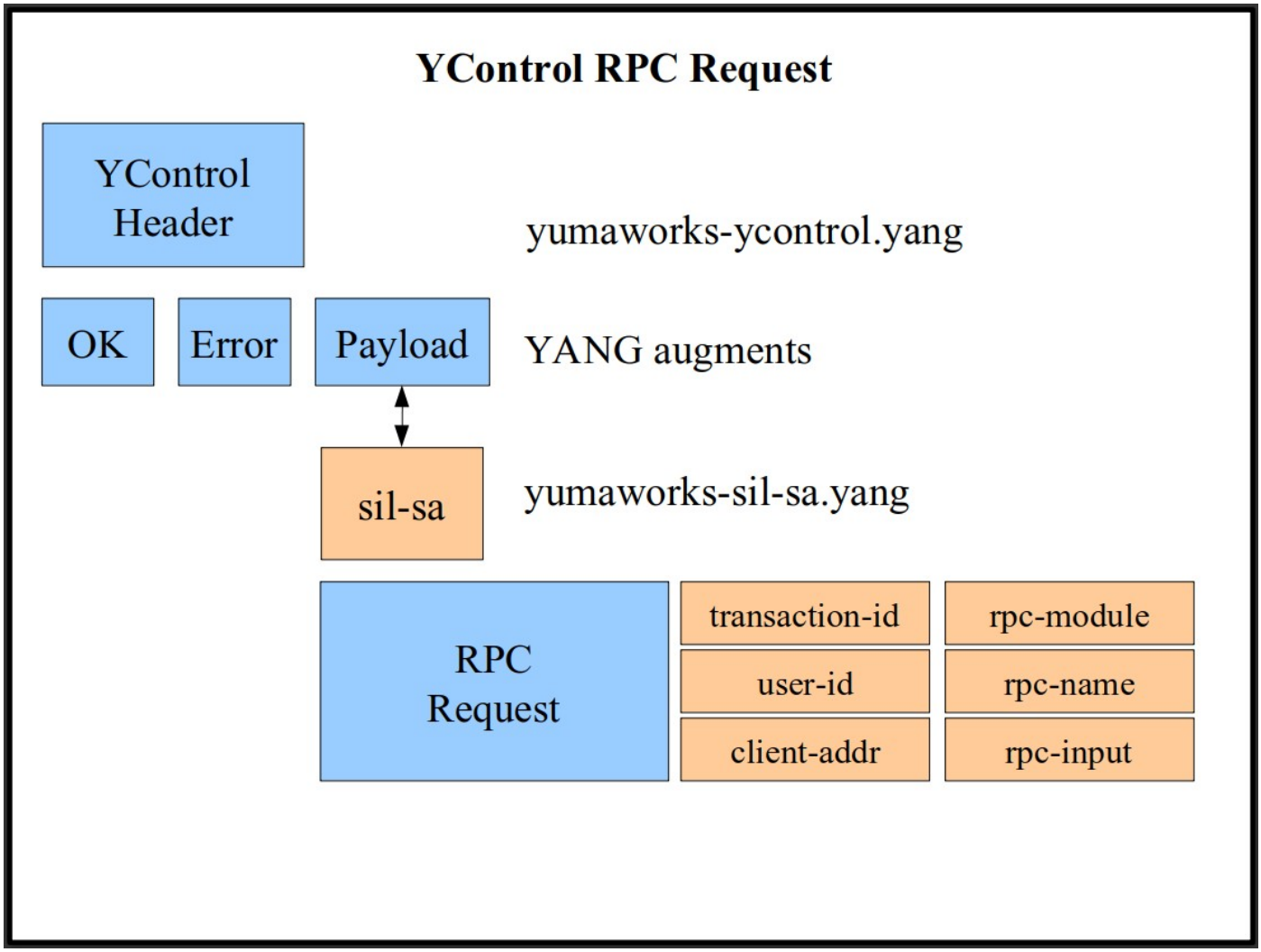
This message requests that a new remote procedure call be validated and invoked on the subsystem.

If there are input parameters the subsystem must validate them.

If not valid or if the operation cannot be performed, the subsystem must return an error.

Expected Response Message: rpc-response

The following diagram illustrates the **<rpc-request>** event message:

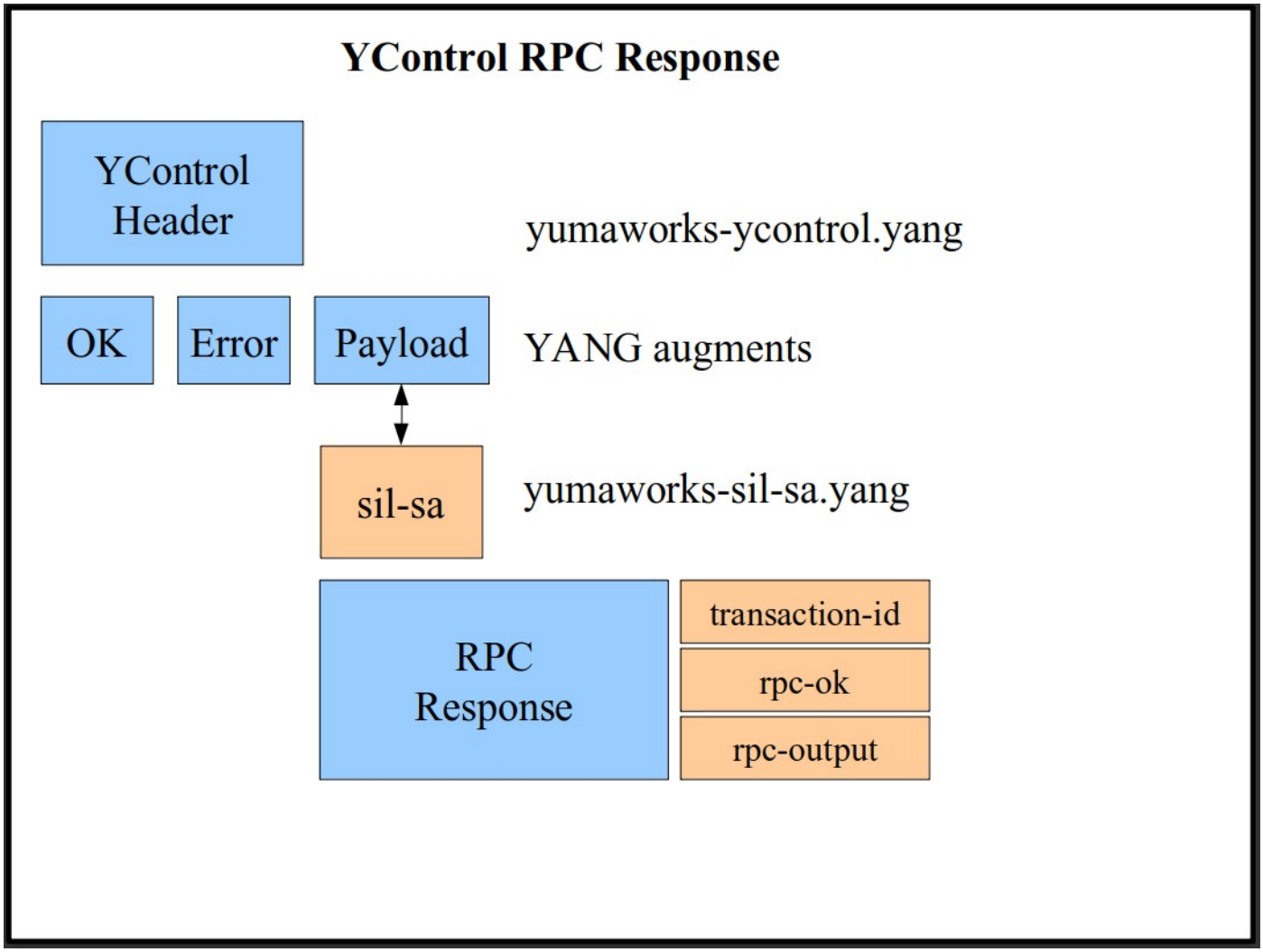


11.2.14 SIL-SA RPC Response Message

The **<rpc-response>** is a subsystem response message. The purpose of the message is to return RPC data or status.

Expected Response Message: none

The following diagram illustrates the **<rpc-response>** event message:



11.2.15 SIL-SA Action Request Message

The **<action-request>** is a server request message. The purpose of the message is to start an ACTION transaction which may require the SIL-SA callback functions on the subsystem to be invoked.

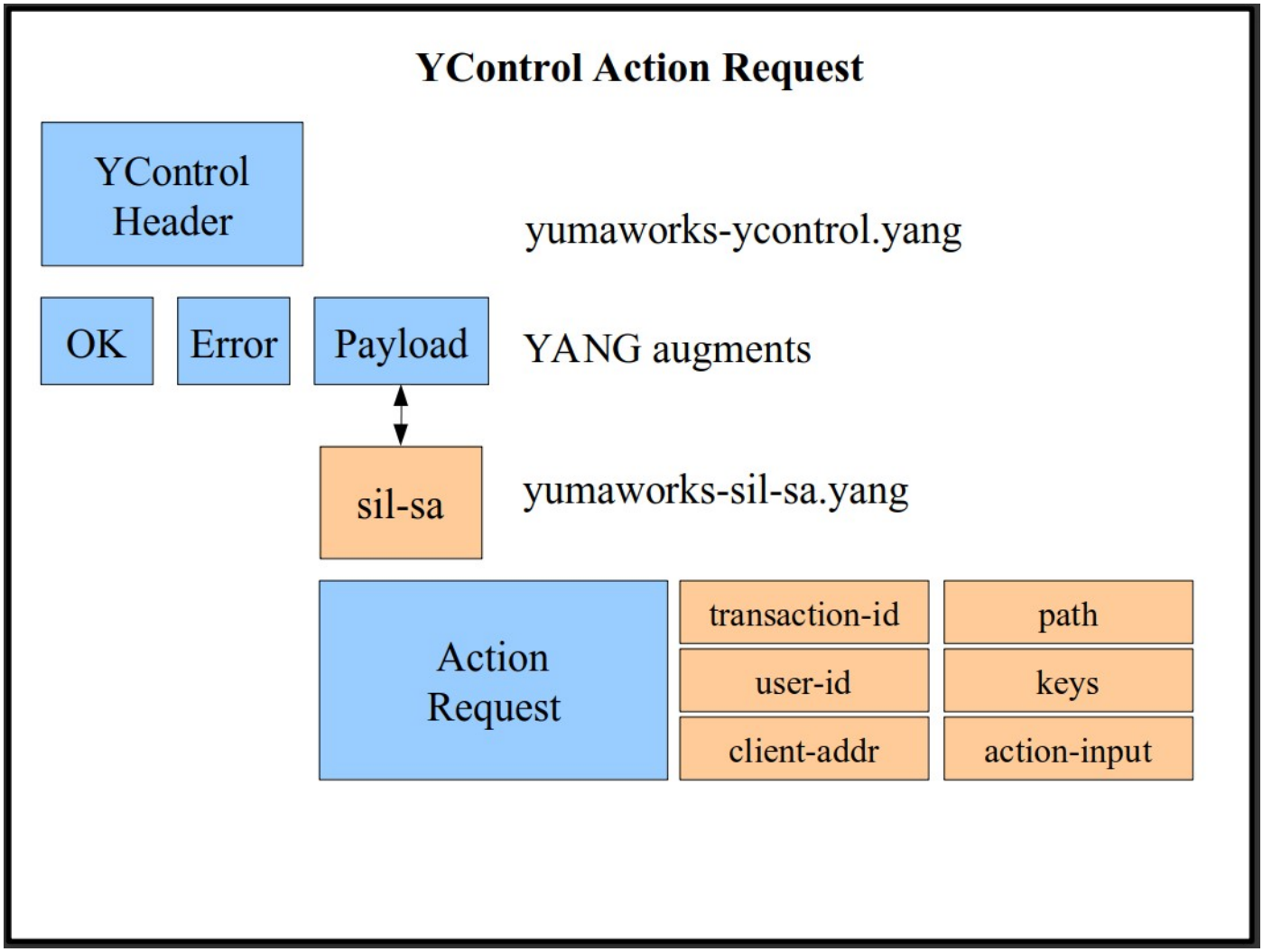
This message requests that a new action call be validated and invoked on the subsystem.

If there are input parameters the subsystem must validate them.

If not valid or if the operation cannot be performed, the subsystem must return an error.

Expected Response Message: action-response

The following diagram illustrates the **<action-request>** event message:

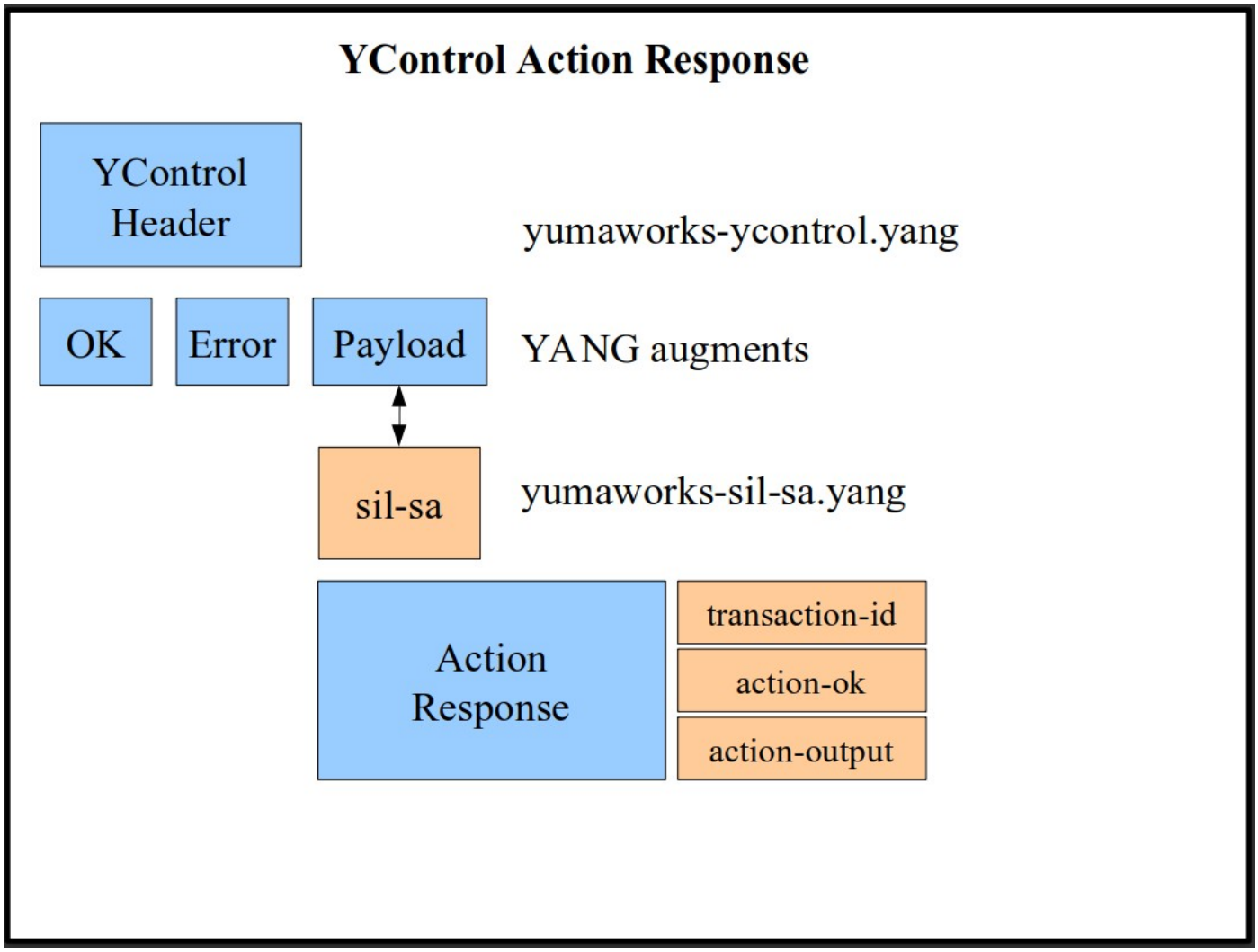


11.2.16 SIL-SA Action Response Message

The **<action-response>** is a subsystem response message. The purpose of the message is to return Action data or status.

Expected Response Message: none

The following diagram illustrates the **<action-response>** event message:



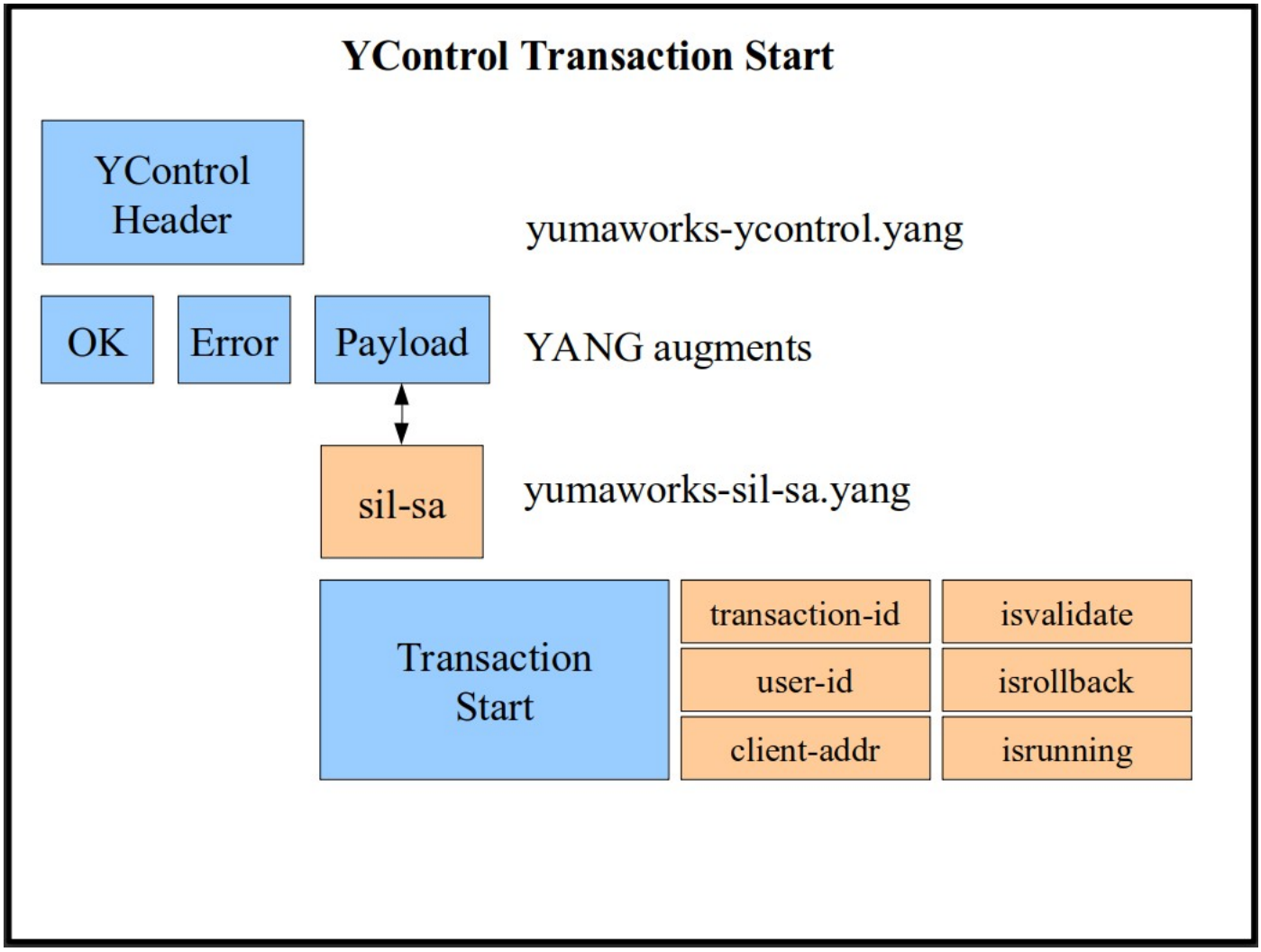
11.2.17 SIL-SA Transaction Start Message

The **<trans-start-hook>** is a server request message. The purpose of the message is to send a request which may require the SIL-SA callback functions on the subsystem to be invoked.

This message requests that Transaction Start Hook callbacks should be invoked on the subsystem.

Expected Response Message: a regular error message or OK.

The following diagram illustrates the **<trans-start-hook>** event message:

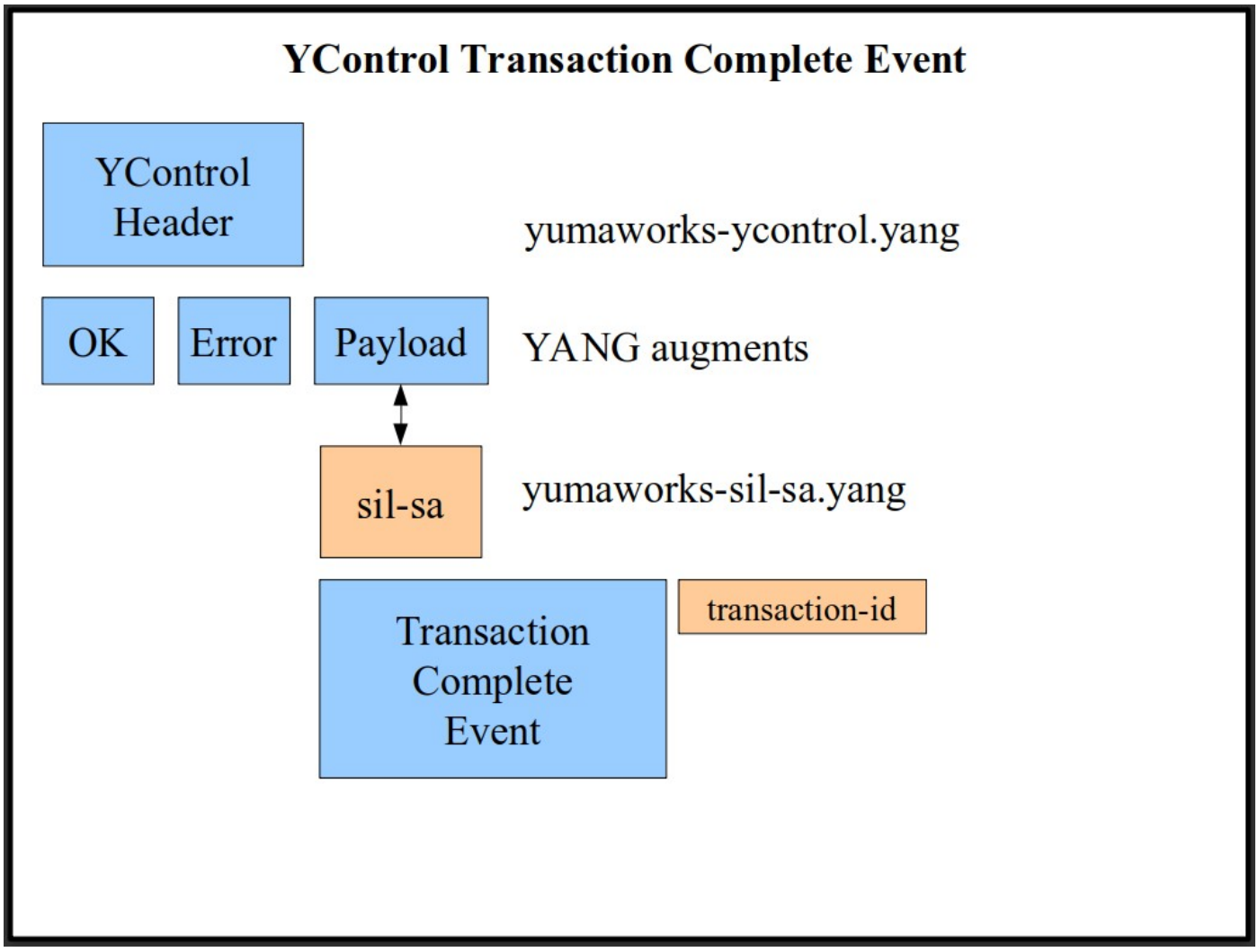


11.2.18 SIL-SA Transaction Complete Event

The **<trans-complete-hook>** is a server event message. The purpose of the message is to send an event which may require the SIL-SA callback functions on the subsystem to be invoked.

Expected Response Message: none

The following diagram illustrates the **<trans-complete-hook>** event message:



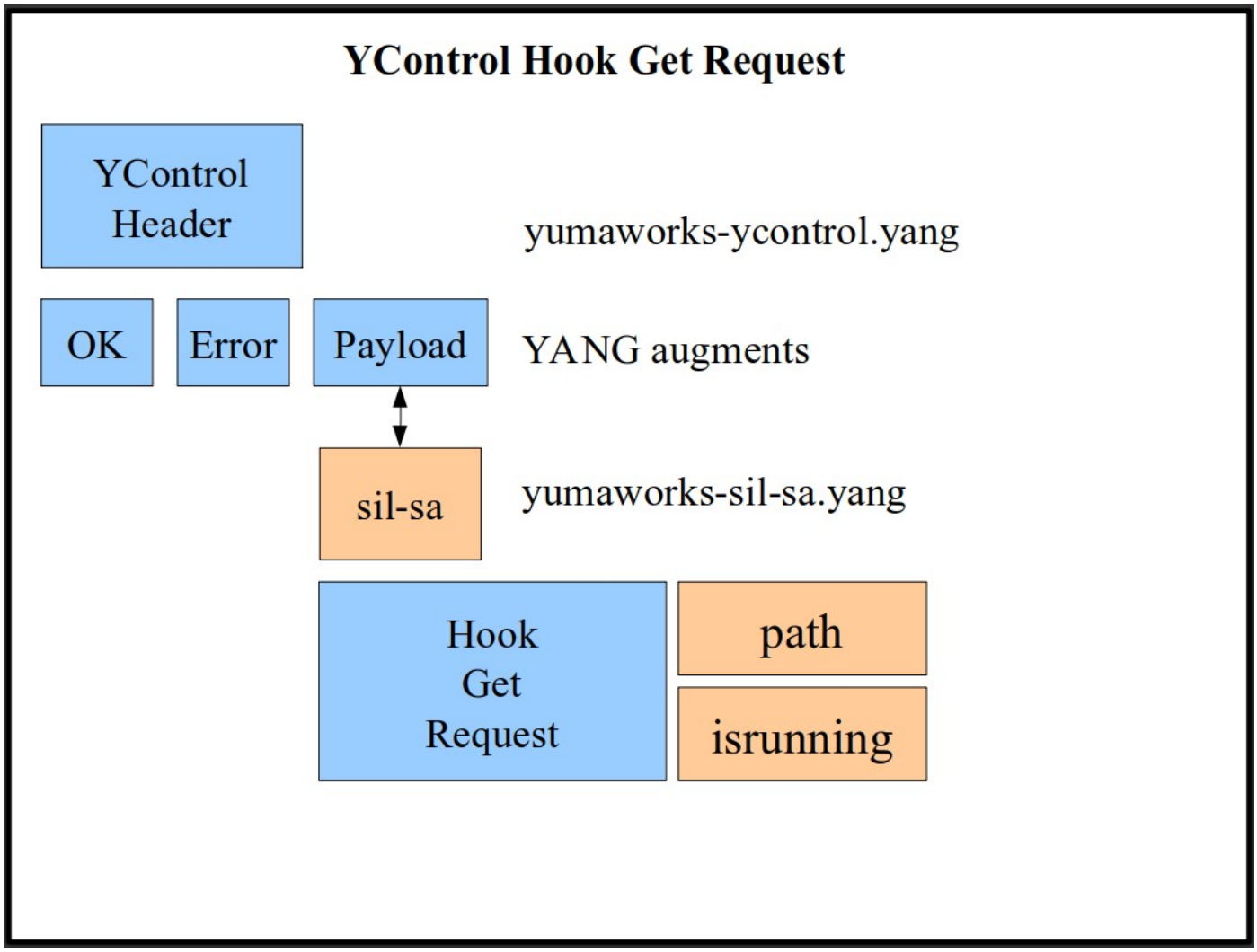
11.2.19 SIL-SA Hook Get Request Message

The **<hook-get-request>** is a subsystem request message. The purpose of the message is to start a transaction which may require the server to run **agt_val_get_data**() API and return result.

Get the **val_value** based on XPath of object instance. This function will return value only if there is existing node in the datastore or there is defaults for the node.

Expected Response Message: hook-get-response

The following diagram illustrates the **<hook-get-request>** event message:

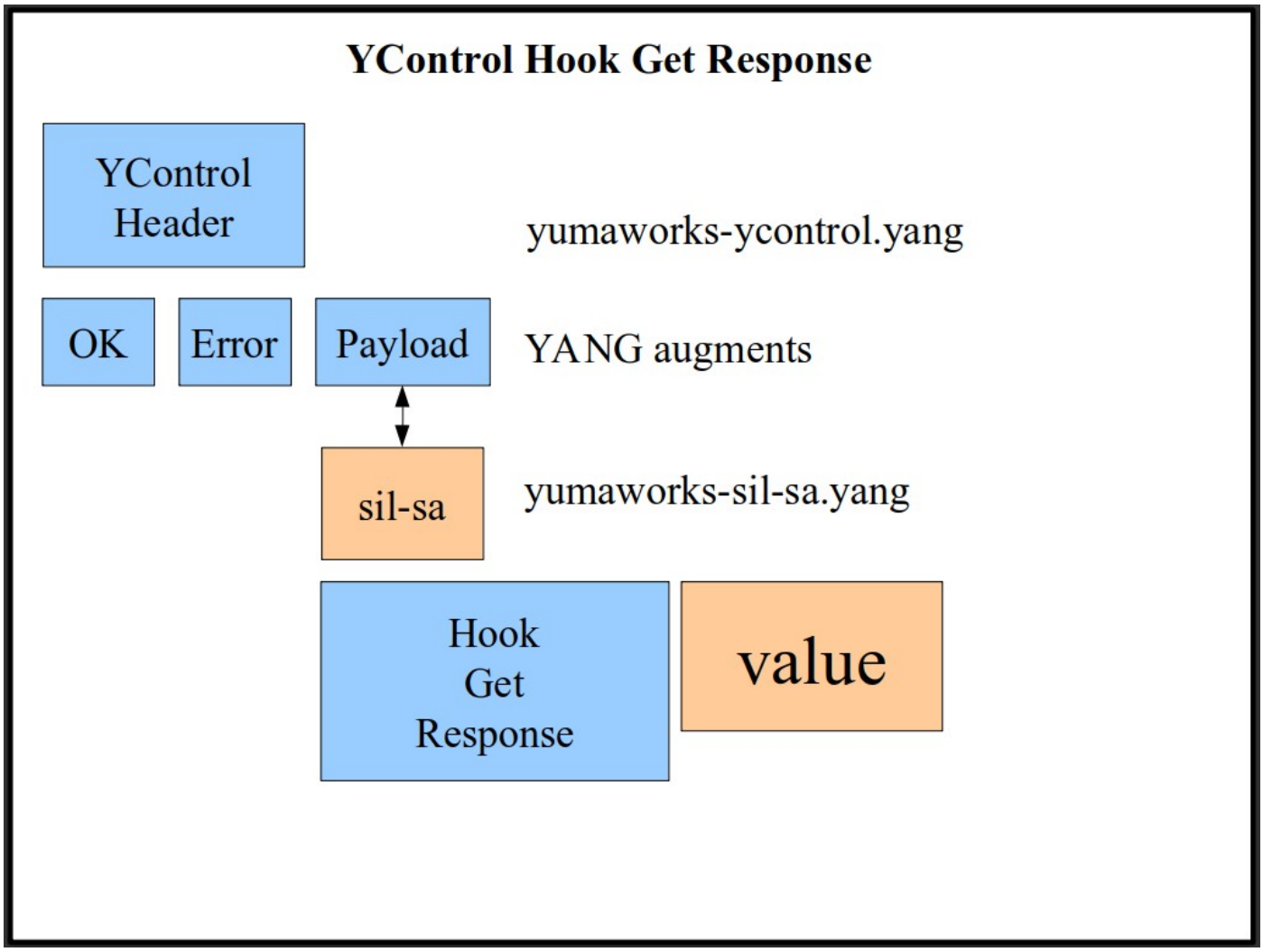


11.2.20 SIL-SA Hook Get Response Message

The **<hook-get-response>** is a server reply message. The purpose of the message is to send the element containing the requested **val_value** based on XPath of the object instance.

Expected Response Message: None

The following diagram illustrates the **<hook-get-response>** event message:

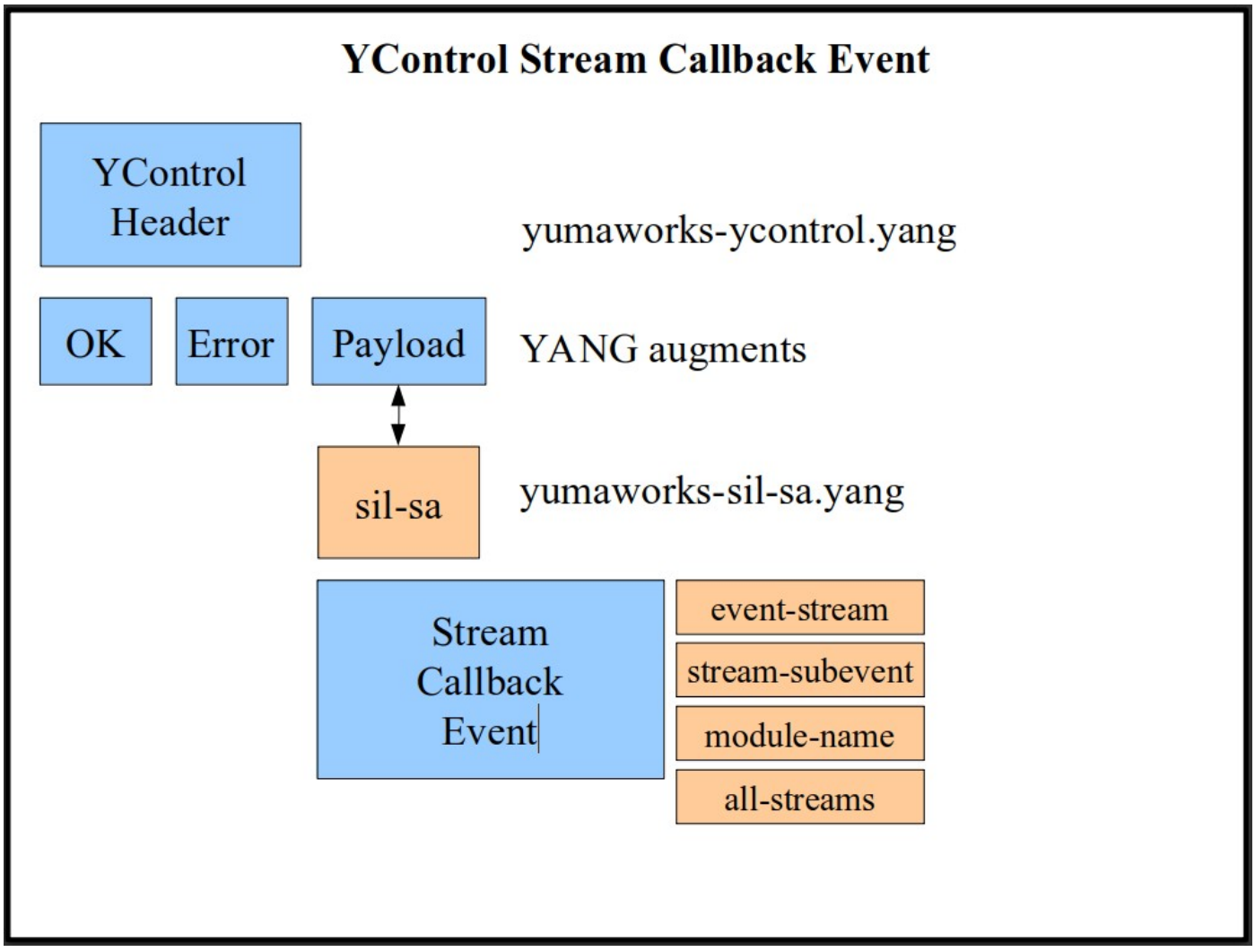


11.2.21 SIL-SA Stream Callback Event

The **<stream-callback-event>** is a server event message. The purpose of the message is to invoke an event-stream callback on a SIL-SA platform. One event will be generated for each remote subsystem entry found that needs to be invoked. E.g., if 3 modules register but all are mapped to the same stream, then 3 events would be sent to the subsystem.

Expected Response Message: None

The following diagram illustrates the **<stream-callback-event>** event message:



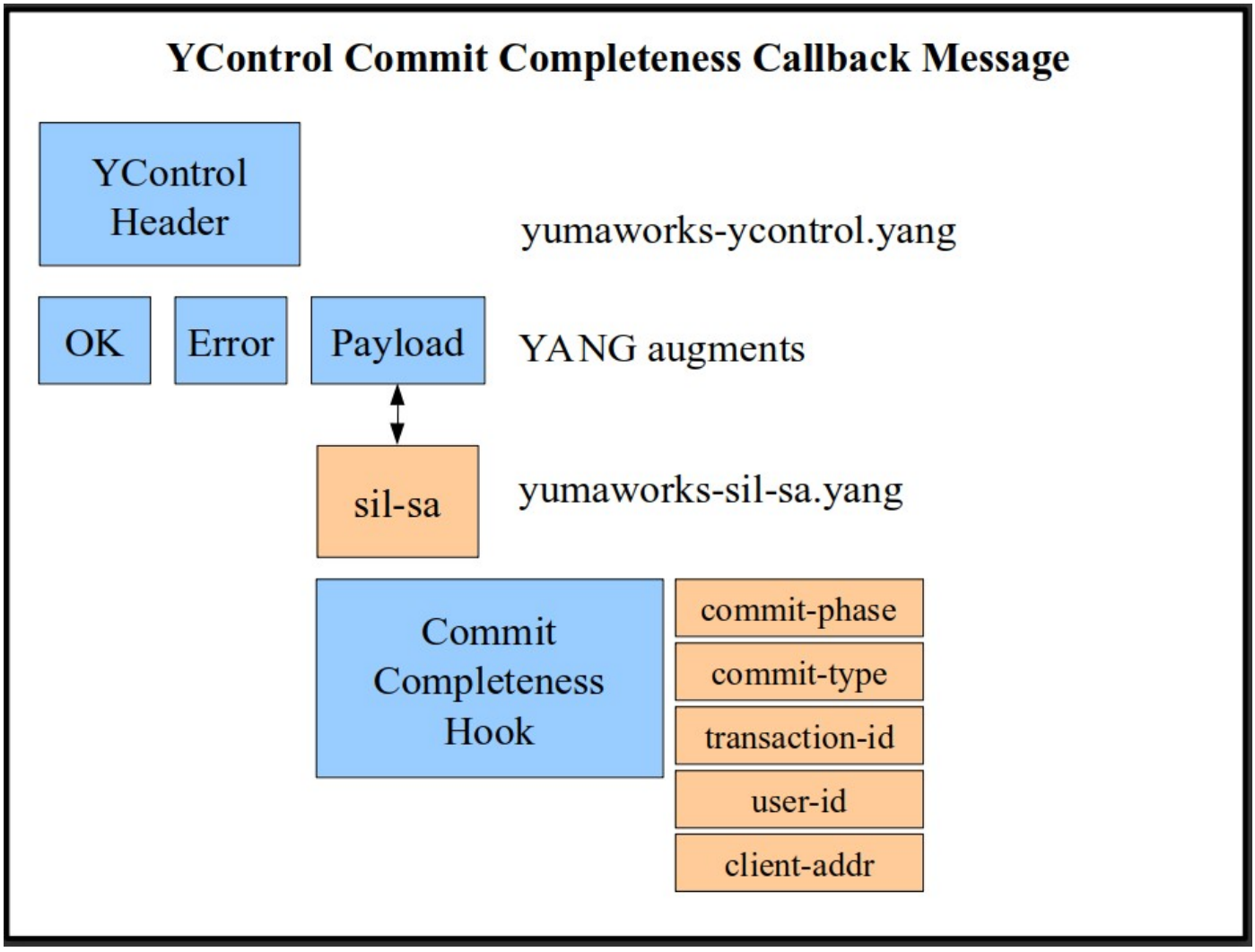
11.2.22 SIL-SA Commit Completeness Hook Message

The **<commit-completeness-hook>** is a server request message. The purpose of the message is to start a transaction which may require the SIL-SA callback functions on the subsystem to be invoked.

This message requests that a **Commit Completeness** callback should be invoked on the subsystem.

Expected Response Message: a regular error message or OK.

The following diagram illustrates the **<commit-completeness-hook>** message:



11.2.23 yumaworks-sil-sa YANG Module

```
module yumaworks-sil-sa {
  yang-version 1.1;
  namespace "http://yumaworks.com/ns/yumaworks-sil-sa";

  prefix "ysil";

  import yuma-types { prefix yt; }
  import yumaworks-types { prefix ywt; }
  import yumaworks-ycontrol { prefix yctl; }
  import yumaworks-agt-profile { prefix yprof; }

  organization "YumaWorks, Inc.";

  contact
    "Support <support at yumaworks.com>";

  description
    "YumaPro SIL Sub-Agent message definitions.

     Copyright (c) 2014 - 2020 YumaWorks, Inc. All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the BSD 3-Clause License
     http://opensource.org/licenses/BSD-3-Clause";

  revision 2021-10-28 {
    description
      "YPW-1857: SIL-SA EDIT2 callback on leaf-list node malfunction.
        Add remove to the enum list of operation;

       YPW-1864: Commit Completeness callbacks support for SIL-SA:
        - Add validate-complete boolean leaf
        - Add apply-complete boolean leaf
        - Add commit-complete boolean leaf
        - Add rollback-complete boolean leaf
        - Add commit-completeness-hook message

       YPW-1870: SIL-SA editvars support
        - Add editvars container to edit list
        - Add editvars container to child-edit list";
  }

  revision 2020-08-27 {
    description
      "Add SIL-SA EDIT2 callbacks support:
        - Add leaf-list edit2-path to register-request message
        - Add child-edit list to edit list in <start-transaction> message";
  }

  revision 2020-06-05 {
    description
      "Add SIL-SA Hook callbacks support:
        - Add leaf-list sethook-path to register-request message
        - Add leaf-list post-sethook-path to register-request message
        - Add leaf-list transaction-path to register-request message
        - Add hook-request message
```

```
       - Add hook-response message
       - Add hook-get-request message
       - Add hook-get-response message";
}

  revision 2019-10-11 {
      description
        "Add with-origin flag for NMDA support";
  }

  revision 2019-08-18 {
      description
        "Add stream-name to notification message.
         Add client-addr to transaction messages to support
         sil_sa_get_client_addr function";
  }

  revision 2019-07-02 {
    description
      "Added HOOKS support:
        * Add following leafs to register-request message:
          - Add transaction-start boolean leaf
          - Add transaction-complete boolean leaf
        * Add trans-start-hook message
        * Add trans-complete-hook message";
  }

  revision 2019-04-20 {
    description
      "Added ACTION support:
        * Add action-path to register-request message
        * Add action-request message
        * Add action-response message";
  }

  revision 2019-01-31 {
      description
        "Added user-id-obj grouping used by start-transaction,
         get-request and rpc-request messages";
  }

  revision 2019-01-20 {
      description
        "Add deviation parm to config-response message.";
  }

  revision 2018-12-05 {
      description
        "Add load-config parameter to start-transaction message";
  }

  revision 2018-10-03 {
      description
        "Add bundle parameter to register-request message";
  }

  revision 2018-06-03 {
      description
        "Add rpc-request and rpc-response messages";
  }

  revision 2017-09-19 {
      description
```

```
        "Fix broken augment statement for payload";
    }

    revision 2015-11-01 {
        description
          "Added support for GETBULK to get-response message.";
    }
    revision 2015-08-17 {
        description
          "Added 'select' node support to get2 SIL-SA get-request
           message";
    }
    revision 2015-04-17 {
        description
          "Updated notification container for release";
    }

    revision 2015-01-15 {
        description
          "Add get and notification support";
    }

    revision 2014-11-18 {
        description
          "Fix module type, supposed to be NcModuleNameSpec
           to allow foo@2000-01-01, not just foo.
           Add bundle-module leaf-list to register msg
           Add bundle-load-event to inform server of any modules
           loaded from a SIL-SA bundle; Remove shutdown-event.";
    }

    revision 2014-09-06 {
        description
          "Add agt-profile container to config-parms grouping
           used in the <config-response> from server to subsystem.";
    }

    revision 2014-08-01 {
        description
          "Initial version.";
    }

    grouping bundle-module-parm {
      leaf-list bundle-module {
        type yt:NcModuleSpec;
        description
          "Module parameters that were loaded as a result of
           any bundle parameters. They will be returned in the
           form modname@revision-date.";
      }
    }

    grouping bundle-parm {
      leaf-list bundle {
        type yt:NcxName;
        description
          "Bundle names that were loaded as a result of
           any bundle parameters.";
      }
    }

    grouping path-parm {
      leaf path {
```

```
      mandatory true;
      type string;
      description
        "Object identifier string:

          * matches 'path' in the register-request message
            for an EDIT callback
          * matches 'get-path' in the register-request message
            for a GET2 callback
          * matches 'action-path' in the register-request message
            for an ACTION callback

         The path string is generated with YANG prefixes using
         the obj_gen_object_id_prefix function in ncx/obj.h
        ";
    }
  }

  grouping system-callbacks-parm {
    description
      "Specifies what system callbacks are enabled.";

    leaf transaction-start {
      type empty;
      description
        "The leaf specifies if there is a Transaction
         Start Callback to register";
    }
    leaf transaction-complete {
      type empty;
      description
        "The leaf specifies if there is a Transaction
         Complete Callback to register";
    }
    leaf validate-complete {
      type empty;
      description
        "The leaf specifies if there is a Validate
         Complete Callback to register";
    }
    leaf apply-complete {
      type empty;
      description
        "The leaf specifies if there is an Apply
         Complete Callback to register";
    }
    leaf commit-complete {
      type empty;
      description
        "The leaf specifies if there is a Commit
         Complete Callback to register";
    }
    leaf rollback-complete {
      type empty;
      description
        "The leaf specifies if there is a Rollback
         Complete Callback to register";
    }
  }

  grouping operation-parms {
    leaf operation {
      mandatory true;
```

```
      type enumeration {
        enum merge;
        enum replace;
        enum create;
        enum delete;
        enum load;
        enum commit;
        enum remove;
      }
      description
        "Operation matching op_editop_t enumeration list.";
    }
  }

  grouping hook-edit-parms {
    leaf operation {
      type string;
      description
        "Operation matching yang_patch_op_t enumeration list.
         Specific for Set Hook add_edit() API";
    }
    leaf where {
      type string;
      description
        "Specifies how a node should be inserted within a
         user-ordered list";
    }
    leaf point {
      type string;
      description
        "Specifies an insertion point for a data resource that
         is being created or moved within a user ordered list
         or leaf-list.";
    }
    leaf skip-callback {
      type boolean;
      description
        "Specifies whether the server should invoke callbacks
         for an added edit";

      default "false";
    }
  }

  grouping hooks-param {
    leaf hook-path {
      type string;
      description
        "Path string for object for Hook callbacks";
    }
    leaf hook-format {
      type enumeration {
        enum node;
        enum subtree;
      }
      default "node";
      description
        "Hook format; dictates specific hook functionality";
    }
    leaf hook-type {
      type enumeration {
        enum none;
        enum set-hook;
```

```
        enum post-set-hook;
        enum transaction-hook;
      }
      description
        "Specifies Hook callback type";
    }
    leaf added-edit {
      type boolean;
      description
        "Specifies whether the edit was added by Set Hook
         of Post Set Hook callbacks. Flag to prevent Set Hook
         looping; will not allow to call Set Hook family callbacks";
    }
  }

  grouping transaction-id-obj {
    leaf transaction-id {
      mandatory true;
      type string;
      description
        "Server specific transaction identifier.";
    }
  }

  grouping user-id-obj {
    leaf user-id {
      type string;
      description
        "Identifies the user that initiated this transaction.";
    }
  }

  grouping client-addr-obj {
    leaf client-addr {
      type string;
      description
        "Identifies the address of the client that initiated
         this transaction.";
    }
  }

  grouping config-parms {
    uses yprof:agt-profile;
    leaf-list bundle {
      type yt:NcxName;
      description "Bundle SIL SA libraries to load";
    }
    leaf-list module {
      type yt:NcModuleSpec;
      description "Module SIL SA libraries to load";
    }
    leaf-list deviation {
      type yt:NcModuleSpec;
      description "Deviations and annotations to load";
    }
  }

  grouping event-stream-parm {
    leaf event-stream {
      type ywt:NcxNumName;
      description
        "Event stream name to use for the operation.";
    }
```

```
  }

  grouping stream-subevent-parm {
    leaf stream-subevent {
      type enumeration {
        enum active {
          description "Event stream is active";
        }
        enum inactive {
          description "Event stream is inactive";
        }
      }
      description
        "Event stream callback subevent types.";
    }
  }

  grouping commit-phase-parm {
    leaf commit-phase {
      type enumeration {
        enum validate {
          description "Invoke Validate Complete callback";
        }
        enum apply {
          description "Invoke Apply Complete callback";
        }
        enum commit {
          description "Invoke Commit Complete callback";
        }
        enum rollback {
          description "Invoke Rollback Complete callback";
        }
      }
      description
        "Specifies the Commit Completeness callback phase.";
    }
  }

  grouping commit-type-parm {
    leaf commit-type {
      type enumeration {
        enum normal {
          description "Commit operation was completed";
        }
        enum reply {
          description "Replay-commit operation was completed";
        }
        enum none {
          description "Type is not set";
        }
      }
      description
        "Specifies the type of commit that was just completed.
         Used only for Commit Complete Callback.";
    }
  }

  grouping return-val {
    anyxml return-keys {
      description
        "List of all ancestor or self key values for the
         object being retrieved, identified by the 'path'
         value. There will be one child leaf for each key
```

```
          in each list.";
    }

    anyxml values {
      description
        "Represents the retrieved values, if any.
         There will be one child node for each returned
         value.";
    }
  }

  grouping editvars-parm {
    container editvars {
      leaf insert-op {
        type enumeration {
          enum none;
          enum first;
          enum last;
          enum before;
          enum after;
        }
        description
          "Specifies the YANG insert operation.";
      }
      leaf insert-str {
        type string;
        description
          "Identifies the saved string for value or key attr.";
      }
      leaf insert-mode {
        type enumeration {
          enum none;
          enum key;
          enum value;
          enum point;
        }
        description
          "Specifies the insert mode requested.";
      }
      anydata insert-val {
        description
          "Represents the insert value, if any.
           Should be present if the insert operation is 'before'
           or 'after'.";
      }
      leaf sil-priority {
        type uint8;
        description
          "Identifies the 2nd SIL priority for server.";
      }
      leaf operset {
        type empty;
        description
          "Specifies whether operation is set or not.";
      }
      leaf move {
        type empty;
        description
          "Specifies whether the insert is for MOVE operation or not.";
      }
      leaf skip-sil-partial {
        type empty;
        description
```

```
          "Specifies whether skip_sil_partial is needed or not.";
      }
    }
  }

  augment "/yctl:ycontrol/yctl:message-payload/yctl:payload/yctl:payload" {
    container sil-sa {

      description
        "Server Instrumentation Library Sub-Agent API Messages

         SIL-SA Protocol Initialization:

         1) subsys sends a <config-request> subsys-request msg
         2) server replies with <config-response> server-response
            msg with the agt_profile data
         3) subsys sends a <register-request> subsys-request msg
            with modules and object path-expressions to register.
            Any bundle and bundle-module info is also sent
         4) server sends <ok> server-response
         5) subsys sends a <trigger-replay> subsys-event;
            if any config for this subsystem, server will send
            a <start-transaction> server-request with a config-replay
            transaction

         SIL-SA Protocol Edit Operation:

         1) the server sends a <start-transaction> server-request msg
            with the edit list
         2) the subsys validates the request and sends an
            <ok> subsys-response (or error)
         3) the server sends a <continue-transaction> for the apply
            phase
         4) the subsys applies the request and sends an
            <ok> subsys-response (or error)
         5) the server sends a <continue-transaction> for the commit
            phase
         6) the subsys commits the request and sends an
            <ok> subsys-response (or error).

         If the server does not get to step (5) because of an error,
         a <continue-transaction> server-request msg will be sent
         for the rollback phase.

         If the server does not get to step (3) because of an error,
         it will send a <cancel-transaction> server-event msg so
         the subsystem can release any saved state.

         RPC Operation:

         1) the server sends an <rpc-request> with the input parameters
         2) the subsystem performs VALIDATE and INVOKE phases
         3) the subsystem returns <rpc-ok> or <rpc-data>.
            YControl error is used for any errors

         ACTION Operation:

         1) the server sends an <action-request> with the input parameters
         2) the subsystem performs VALIDATE and INVOKE phases
         3) the subsystem returns <action-ok> or <action-data>.
            YControl error is used for any errors

         HOOKS Support:
```

```
          Transaction Start Hooks:
             1) the server send trans-start-hook request
             2) the subsys validates the request, invoke callbacks, and
                sends an <ok> subsys-response (or error)

          Transaction Complete Hooks:
             1) the server send trans-complete-hook event msg so subsystem
                can invoke Transaction Complete Callbacks
        ";

      choice message-type {
        mandatory true;

        leaf config-request {
          type empty;
          description
           "Message type: subsys-request;
            Purpose: register the service with the server
            and request the service configuration from server.
            Expected Response Message: config-response";
        }

        container config-response {
          description
           "Message type: server-reply;
            Purpose: server will send this element containing the
            requested sub-system configuration.
            Expected Response Message: none";

          uses config-parms;
        }

        container register-request {
          description
           "Message type: subsys-request;
            Purpose: register the SIL-SA callback functions
            for this sub-system.
            Expected Response Message: ok";

          uses bundle-parm;
          uses bundle-module-parm;

          list register {
            description
              "Specifies the path strings of all the objects
               in each module that is supported by the subsystem.";

            key "module";
            leaf module {
              type yt:NcxName;
              description
                "Module name for the target object.";
            }
            leaf-list path {
              type string;
              description
                "Path string for object for an EDIT callback";
            }
            leaf-list edit2-path {
              type string;
              description
                "Path string for object for an EDIT2 callback";
```

```
        }
        leaf-list get-path {
          type string;
          description
            "Path string for object for a GET callback";
        }
        leaf-list rpc-name {
          type yt:NcxName;
          description
            "Name of the RPC operation callback";
        }
        leaf-list action-path {
          type string;
          description
            "Path string for object for an ACTION callback";
        }

        leaf-list post-sethook-path {
          type string;
          description
            "Path string for object for a Post Set Hook callback";
        }
        list sethook-list {
          description
            "List for a Set Hook callbacks";

          key "hook-path";
          uses hooks-param;
        }
        list transaction-hook-list {
          description
            "List for a Transaction Hook callbacks";

          key "hook-path";
          uses hooks-param;
        }

        container stream-callback {
          presence
           "Indicates a stream-callback is registered for
            the 'module' in the 'register' list entry with
            the 'register-request' message. If no child nodes
            then the registration is for the module name.
            Otherwise it is for one stream or all streams.";

          choice callback-type {
            case event-stream {
              uses event-stream-parm;
            }

            leaf all-streams {
              type empty;
              description
                "Flag to register for all event-stream events";
            }
          }
        }
      }

      uses system-callbacks-parm;
    }

    container start-transaction {
```

```
            description
             "Message type: server-request;
              Purpose: Start an edit transaction which may require the
              SIL-SA callback functions on the subsystem to be invoked.

              This message requests that a new edit transaction be
              started on the subsystem.  Only 1 transaction can be in
              progress at a time.

              If this transaction is for a validate operation then
              there will not be any followup messages.  Otherwise,
              the subsystem will retain this message until a
              cancel-transaction message has been received with the
              same transaction-id value, or a continue-transaction
              message has been received with the same transaction-id
              value for the 'rollback' or 'commit' phase.

              Expected Response Message: transaction-response or error";

            uses transaction-id-obj;

            uses user-id-obj {
              refine "user-id" {
                mandatory true;
              }
            }
            uses client-addr-obj;

            leaf target {
              mandatory true;
              type string;
              description
                "Identifies the target datastore being edited.
                 The values 'candidate' and 'running' are supported.";
            }

            leaf validate {
              type boolean;
              default false;
              description
                "If 'true' then this start-transaction is for a validate
                 operation or edit that is not on the running
                 configuration datastore, so there will not be
                 any followup messages at all for this message.
                 The subsystem will release this info instead of caching it,
                 and not expect any more messages for the same value of
                 'transaction-id'.

                 If 'false' then this is a normal edit operation and
                 the apply and commit/rollback followup messages
                 will be sent. The subsystem will cache this data
                 until the transaction is cancelled or completed.";
            }

            leaf reverse-edit {
              type boolean;
              default false;
              description
                "If 'true' then this start-transaction is for a
                 reverse-edit operation.  All the phases should
                 be invoked in sequence for the provided edit-list.
                 The transaction can be discarded after sending
                 a response, like the 'validate=true' flag.
```

```
        If 'false' then this is a normal edit operation.";
    }

    leaf load-config {
      type boolean;
      default false;
      description
        "If 'true' then this start-transaction is for a
         <trigger-replay> operation. If 'false' then this is
         a normal edit operation, probably caused by
         a client <edit-config> operation.";
    }

    leaf is-hook-load {
      description
       "Specifies whether this is a LOAD operation or not";

      type boolean;
    }

    leaf is-hook-validate {
      description
       "Specifies whether this is a Validate operation or not";

      type boolean;
    }

    list edit {
      key "id";

      leaf id {
        type uint32;
        description "Arbitrary edit identifier.";
      }

      uses operation-parms;
      uses path-parm;
      uses hooks-param;
      uses editvars-parm;

      anyxml newval {
        description
          "Represents the new value, if any.
           Should be present if operation is 'merge'
           'replace', 'create', or 'load'.";
      }

      anyxml curval {
        description
          "Represents the current value, if any.
           Should be present if operation is 'replace',
           'delete', or 'commit'";
      }

      anyxml keys {
        description
          "List of all ancestor or self key values for the
           object being edited, identified by the 'path' value.
           There will be one child leaf for each key in each list.";
      }

      list child-edit {
        description
```

```
             "List of EDIT2 children edits, only when the operation
              is MERGE";

            uses operation-parms;
            uses editvars-parm;

            anyxml newval {
              description
                "Represents the new child value, if any.
                 Should be present if operation is 'merge'
                 'replace', 'create', or 'load'.";
            }

            anyxml curval {
              description
                "Represents the current child value, if any.
                 Should be present if operation is 'replace',
                 'delete', or 'commit'";
            }
          }  // list child edit

        }  // list edit
      }  // container start-transaction


      container continue-transaction {
        description
         "Message type: server-request;
          Purpose: Invoke a callback phase for an edit transaction
          in progress.


          Expected Response Message: transaction-response or error";

        uses transaction-id-obj;

        leaf phase {
          mandatory true;
          type enumeration {
            enum apply {
              description
                "Apply the curent transaction.
                 Resources can be reserved that will be
                 used in the commit phase.";
            }
            enum commit {
              description
                "Commit the current transaction.";
            }
            enum rollback {
              description
                "Rollback the current transaction.";
            }
          }
          description
            "The SIL-SA callback phase in progress.";
        }
      }  // container continue-transaction


      container transaction-response {
        description
         "Message type: subsys-response
```

```
      Purpose:
        - Set Hook: return added edits data or status
        - Post Set Hook: return added edits data or status
        - Transaction Hook: Expected Response Message: ok or error
        - If no Hook invoked: Expected Response Message: ok or error";

    uses transaction-id-obj;

    list added-edit {
      // no key!! The paths for added edits may be the same

      uses path-parm;
      uses hook-edit-parms;

      anyxml edit {
        description
          "Represents the added edits values, if any.
           There will be one child node for each added edit
           value.";
      }
    }
  }


  container cancel-transaction {
    description
     "Message type: server-event;
      Purpose: Cancel an edit transaction in progress.
      Expected Response Message: none";

    uses transaction-id-obj;

  }  // container cancel-transaction


  leaf trigger-replay {
    type empty;
    description
     "Message type: subsys-event;
      Purpose: Trigger a configueration replay to load
         the running config data into the SIL-SA
         instrumentation.
      Expected Response Message: none; server will send
         a <start-transaction> if there is any
         config for the SIL-SA functions registered
         by the subsystem.";
  }  // leaf trigger-replay


  container load-event {
    description
     "Message type: server-event;
      Purpose: A module or bundle has been loaded or
         unloaded at run-time. Subsys wil load SIL-SA code.
         and trigger a register event for any SIL calls
         registered.
      Expected Response Message: none";

    leaf load {
      type boolean;
      default true;
      description
        "Set to 'true' if this is a load event.
```

```
             Set to 'false' if this is an unload event.";
      }

      uses config-parms;
    }  // container load-event


    container bundle-load-event {
      description
       "Message type: subsys-event;
        Purpose: A SIL-SA bundle has been loaded with
         a load-event sent from the server.  This has
         caused some modules to be loaded on the subsystem,
         that need to be reported back to the main server
         so the datastore validation, agt_state, and other
         system book-keeping can be done.
        Expected Response Message: none";

      uses bundle-parm;
      uses bundle-module-parm;

    }  // container bundle-load-event


    container get-request {
      description
        "Composite retrieval request to support NETCONF
         and RESTCONF get operations.
         Type: server-request
         Expected Response Message: subsys-response
                              (get-response or error)";

      uses transaction-id-obj;
      uses user-id-obj;
      uses client-addr-obj;

      leaf flags {
        type bits {
          bit keys {
            description
              "Return only the key values";
          }
          bit config {
            description
              "Return config=true data nodes";
          }
          bit oper {
            description
              "Return config=false data nodes";
          }
          bit getnext {
            description
              "This is a get-next request instead of a
              get request";
          }
          bit withdef {
            description
              "Return default values for missing nodes";
          }
          bit select {
            description
              "Return only the select nodes and any key leafs.
               Ignore the config, oper, withdef flags if this
```

```
            bit is set.";
        }
        bit with-origin {
          description
            "This is a <get-data> operation and the
             with-origin parameter is selected.";
        }
      }
      default "";
      description
        "Set of get request modifier flags";
    }

    leaf max-entries {
      type uint32;
      description
        "Max number of entries requested.
         The default is '0' == all for leaf-list and
         '1' for all other node types.";
    }

    uses path-parm;

    anyxml keys {
      description
        "List of all ancestor or self key values for the
         object being retrieved, identified by the 'path'
         value. There will be one child leaf for each key
         in each list.";
    }

    anyxml matches {
      description
        "Represents any content-match child leafs for the
         request. All leafs in this container must match the
         corresponding child nodes in an instance of the
         requested list or container, for that instance to
         be returned.

         Any content-match nodes must match in addition
         to any key leafs specified in the 'keys' container.";
    }

    container select-nodes {
      list select-node {
        description
          "Only requesting these nodes be returned. If no
           entries and the 'select' bit is set in the flags
           leaf, then no objects except list keys are
           returned.";

        key objname;
        leaf objname {
          type string;
          description
            "Object name of the select node";
        }
        leaf modname {
          type string;
          description
            "Module name of the select node; If missing then
             use the module-name of the path target object.";
        }
```

```
      }
    }
  }

  container get-response {
    description
      "Composite retrieval response to support NETCONF
       and RESTCONF get operations.
       Type: subsys-response
       Expected Response Message: none";

    uses transaction-id-obj;

    leaf more-data {
      type boolean;
      default false;
      description
        "Indicates if the GET callback has more data to send";
    }

    leaf match-test-done {
      type boolean;
      default false;
      description
        "Indicates if the requested content-match tests have
         be performed. Ignored if the 'matches' parameter
         is missing or empty.";
    }

    leaf active-case-modname {
      type string;
      description
        "Module name of the active case if there is one.
         Only applies if the GET2 object callback is for
         a YANG choice-stmt.";
    }

    leaf active-case {
      type string;
      description
        "Name of the active case if there is one.
         Only applies if the GET2 object callback is for
         a YANG choice-stmt.";

    }

    leaf origin {
      type string;
      description
        "The NMDA origin value for this object.
         Only applies for config=true list and
         presence containers.";
    }

    choice return-choice {
      case return-one {
        description
          "For all objects except YANG list, one entry
           will be returned.  This can also be used
           for YANG list, except in GETBULK mode.";
        uses return-val;
      }
      case return-getbulk {
```

```
        description
          "For YANG list GETBULK mode.  There will be one entry
           for each list instance that met the search criteria.
           If the max_entries parameter was greater than zero,
           there the number of instances of 'entry' should not
           exceed this value.";
        list entry {
          // no key!!
          uses return-val;
        }
      }
    }
  }

  container notification {
    description
      "Subsystem generated YANG notification event
       for NETCONF and RESTCONF streams.
       Type: subsys-event
       Expected Response Message: none";

    leaf module-name {
      mandatory true;
      type string;
      description
        "Module name containing the notification definition";
    }

    leaf event-name {
      mandatory true;
      type string;
      description
        "Notification statement name";
    }

    leaf event-time {
      mandatory true;
      type string;
      description
        "Notification creation timestamp";
    }

    leaf stream-name {
      type yt:NcxName;
      default "NETCONF";
      description
        "Stream name for this notification";
    }

    anyxml parms {
      description
        "List of all parameters that this notification
         is sending in the payload.";
    }
  }

  container rpc-request {
    description
     "Message type: server-request;
      Purpose: Start an RPC transaction which may require the
      SIL-SA callback functions on the subsystem to be invoked.

      This message requests that a new remote procedure
```

```
              call be validated and invoked on the subsystem.

              If there are input parameters the subsystem
              must validate them.

              If not valid or if the operation cannot be performed,
              the subsystem must return an error.

              Expected Response Message: rpc-response";

            uses transaction-id-obj;
            uses user-id-obj;
            uses client-addr-obj;

            leaf rpc-module {
              type yt:NcxName;
              mandatory true;
              description
                "Identifies the module name of the RPC definition.";
            }

            leaf rpc-name {
              type yt:NcxName;
              mandatory true;
              description
                "Identifies the name of the RPC definition.";
            }

            anyxml rpc-input {
              description
                "Contains the RPC input data (if any).";
            }
          }

          container rpc-response {
            description
             "Message type: subsys-response
              Purpose: Return RPC data or status
              Expected Response Message: none";

            uses transaction-id-obj;

            choice response-type {
              leaf rpc-ok {
                type empty;
                description
                  "RPC successfully invoked";
              }
              anyxml rpc-output {
                description
                  "Contains the RPC output data (if any).";
              }
            }
          }

          container action-request {
            description
             "Message type: server-request;
              Purpose: Start an ACTION transaction which may require the
              SIL-SA callback functions on the subsystem to be invoked.

              This message requests that a new action call be validated
              and invoked on the subsystem.
```

```
            If there are input parameters the subsystem
            must validate them.

            If not valid or if the operation cannot be performed,
            the subsystem must return an error.

            Expected Response Message: action-response";

          uses transaction-id-obj;
          uses user-id-obj;
          uses client-addr-obj;
          uses path-parm;

          anyxml keys {
            description
              "List of all ancestor key values for the
               action being invoked, identified by the 'path' value.
               There will be one child leaf for each key in each list.";
          }

          anyxml action-input {
            description
              "Contains the ACTION input data (if any).";
          }
        }

        container action-response {
          description
           "Message type: subsys-response
            Purpose: Return ACTION data or status
            Expected Response Message: none";

          uses transaction-id-obj;

          choice response-type {
            leaf action-ok {
              type empty;
              description
                "ACTION successfully invoked";
            }
            anyxml action-output {
              description
                "Contains the ACTION output data (if any).";
            }
          }
        }

        container trans-start-hook {
          description
           "Message type: server-request;
            Purpose: Send a request which may require the
            SIL-SA callback functions on the subsystem to be invoked;
            Expected Response Message: <ok> or <error>.

            This message requests that Transaction Start Hook callbacks
            should be invoked on the subsystem.";

          uses transaction-id-obj;
          uses user-id-obj;
          uses client-addr-obj;

          leaf isvalidate {
```

```
            description
             "Specifies whether this is a <validate> operation or not.
              TRUE if this is a validate operation transaction";

            type boolean;
          }

          leaf isrollback {
            description
             "Specifies whether this is a confirmed commit timeout
              or a cancel-confirmed-commit operation.
              FALSE if this is some other type of transaction";

            type boolean;
          }

          leaf isrunning {
            description
             "Specifies whether this is transaction for running datastore.
              TRUE if this is a running datastore transaction";

            type boolean;
          }
        }

        container trans-complete-hook {
          description
           "Message type: server-event;
            Purpose: Send an event which may require the
              SIL-SA callback functions on the subsystem to be invoked;
            Expected Response Message: none";

          uses transaction-id-obj;
        }

        container hook-get-request {
          description
           "Message type: subsys-request;
            Purpose: Start a transaction which may require the
            server to run agt_val_get_data() API and return result.

            Get the val_value based on XPath of object instance
            This function will return value only if there is existing node
            in the datastore or there is defaults for the node.

            Expected Response Message: hook-get-response";

          uses path-parm;

          leaf isrunning {
            description
             "Specifies whether to retrieve a value from the running
              datastore or candidate";

            type boolean;
          }
        }

        container hook-get-response {
          description
           "Message type: server-reply;
            Purpose: server will send this element containing the
              requested val_value based on XPath of object instance.
```

```
              Expected Response Message: none";

            anyxml value {
              description
                "Represents the the val_value based on XPath of
                 object instance";
              }
            }

          container stream-callback-event {
            description
             "Message type: server-event;
              Purpose: An event-stream callback on a SIL-SA platform
                 needs to be invoked.  One event will be generated
                 for each remote subsystem entry found that needs
                 to be invoked. E.g., if 3 modules register but all are
                 mapped to the same stream, then 3 events would be
                 sent to the subsystem.
              Expected Response Message: none";

            uses event-stream-parm;

            uses stream-subevent-parm;

            // info to identify the SIL-SA callback to invoke
            leaf module-name {
              type yt:NcxName;
                description
                  "Module name to use to identify callback.";
              }

            leaf all-streams {
              type empty;
               description
                 "All streams flag to identify callback.";
              }
            }  // container stream-callback-event


          container commit-completeness-hook {
            description
             "Message type: server-request;
              Purpose: Send a request which may require the
                SIL-SA callback functions on the subsystem to be invoked;

                This message requests that Commit Completeness callbacks
                should be invoked on the subsystem.

              Expected Response Message: <ok> or <error>.";

            uses transaction-id-obj;
            uses user-id-obj;
            uses client-addr-obj;

            uses commit-phase-parm;
            uses commit-type-parm;
          } // container commit-completeness-hook

        } // choice message-type
      } // container sil-sa
    } // augment
}
```

12 DB-API Subsystem

The DB-API subsystem provides a special datastore editing interface for subsystems to access the YANG-based data controlled by the main server. It can be used by network management components within the device that operate in addition to the **netconfd-pro** server.

The DB-API service provides the ability for a subsystem to send a configuration edit to the main server. The YControl protocol is used to manage the connection and communication with the main server.

The db-api-app program "**main.c**" function shows an example of how the DB-API interface can be used to send an edit request to the server. It also shows how db-lock feature can be utilized.

Example DB-API Application

```
/********************************************************************
* FUNCTION check_db_lock_test
*
* Check elapsed time to simulate local-lock and local-unlock
* operations to cause the server to get db-lock failures
*
*********************************************************************/
static void
    check_db_lock_test (uint32 st_time, uint32 change_time, boolean *done)
{
    static int32 lock_count = 0; // restrict infinite loop
    time_t timenow;
    (void)time(&timenow);

    //uint32 start_wait = 30;   // wait this long before anything
    //uint32 change_wait = 7;   // wait to change lock state

    double timediff = difftime(timenow, start_time);

    if ( lock_count > 50) {
        *done = TRUE ;
        log_info("\nTest Completed!\n");
        return;
    }

    if (timediff < (double)st_time) {
        return;
    }

    status_t res = NO_ERR;
    if (last_time == 0) {
        /* request first local lock */
        res = db_api_request_local_db_lock();
        if (res == NO_ERR) {
            log_info("\ndb-lock-test: Got local db-lock");
            locked = TRUE;
            (void)time(&last_time);
        } else {
            log_info("\ndb-lock-test: First local-lock failed (%s)",
                     get_error_string(res));
        }
        lock_count++;
    } else {
        timediff = difftime(timenow, last_time);
        if (timediff < (double)change_time) {
```

```
            return;
        }

        if (locked) {
            res = db_api_release_local_db_lock();
            if (res == NO_ERR) {
                log_info("\ndb-lock-test: Released local db-lock");
                locked = FALSE;
                last_time = timenow;   // reset timer
            } else {
                log_info("\ndb-lock-test: Release local-lock failed (%s)",
                         get_error_string(res));
            }
        } else {
            res = db_api_request_local_db_lock();
            if (res == NO_ERR) {
                log_info("\ndb-lock-test: Request local db-lock");
                locked = TRUE;
                last_time = timenow;
            } else {
                log_info("\ndb-lock-test: Request local-lock failed (%s)",
                         get_error_string(res));
            }
        }
        lock_count++ ;
    }

}  /* check_db_lock_test */


/********************************************************************
* FUNCTION send_test_edit
*
* This is an example send edit function.
*
*********************************************************************/
static void
    send_test_edit (void)
{
    /* simple leaf test from test.yang */
    const xmlChar *path_str = (const xmlChar *)"/int8.1";
    const xmlChar *operation_str = (const xmlChar *)"merge";
    const xmlChar *value_str = (const xmlChar *)"<int8.1>42</int8.1>";

    status_t res = db_api_send_edit(path_str, operation_str, value_str);
    if (res != NO_ERR) {
        log_error("\nSend test edit failed %s %s = %s (%s)\n",
                  operation_str, path_str, value_str,
                  get_error_string(res));
    } else if (LOGDEBUG) {
        log_debug("\nSend test edit OK  %s %s = %s\n",
                  operation_str, path_str, value_str);
    }

}  /* send_test_edit */


/********************************************************************
* FUNCTION send_complex_test_edit
*
* This is an example send edit function.
* It uses multiple APIs to build a patch with 3 edits
* The module test.yang needs to be loaded for this to work
```

```
*****************************************************************/
static void
    send_complex_test_edit (void)
{
    const xmlChar *patch_id_str = (const xmlChar *)"complex-P2";
    boolean system_edit = TRUE;
    yang_patch_cb_t *pcb = NULL;

    /* Step 1: create a patch control block */
    status_t res =
        db_api_start_patch(patch_id_str,
                           system_edit,
                           &pcb);

    if (pcb == NULL) {
        log_error("\nCreate patch failed (%s)\n", get_error_string(res));
        return;
    }

    const xmlChar *edit_id_str = (const xmlChar *)"edit1";
    const xmlChar *edit_target_str = (const xmlChar *)"/int8.1";
    const xmlChar *edit_operation_str = (const xmlChar *)"replace";
    const xmlChar *edit_xml_value = (const xmlChar *)
        "<int8.1 xmlns='http://netconfcentral.org/ns/test'>44</int8.1>";
    const xmlChar *insert_point = NULL;
    const xmlChar *insert_where = NULL;

    /* Step 2: add edit1 */
    res = db_api_add_edit(pcb,
                          edit_id_str,
                          edit_target_str,
                          edit_operation_str,
                          edit_xml_value,
                          insert_point,
                          insert_where);
    if (res != NO_ERR) {
        log_error("\nAdd edit1 failed for complex test edit (%s)\n",
                  get_error_string(res));
        db_api_free_patch(pcb);
        return;
    }

    /* Step 3: add edit2 */
    edit_id_str = (const xmlChar *)"edit2";
    edit_target_str = (const xmlChar *)"/int16.1";
    edit_operation_str = (const xmlChar *)"remove";
    edit_xml_value = NULL;

    res = db_api_add_edit(pcb,
                          edit_id_str,
                          edit_target_str,
                          edit_operation_str,
                          edit_xml_value,
                          insert_point,
                          insert_where);
    if (res != NO_ERR) {
        log_error("\nAdd edit2 failed for complex test edit (%s)\n",
                  get_error_string(res));
        db_api_free_patch(pcb);
        return;
    }

    /* Step 4: add edit3 */
```

```
    edit_id_str = (const xmlChar *)"edit3";
    edit_target_str = (const xmlChar *)"/uint32.1";
    edit_operation_str = (const xmlChar *)"create";
    edit_xml_value = (const xmlChar *)
        "<uint32.1 xmlns='http://netconfcentral.org/ns/test'>400</uint32.1>";

    res = db_api_add_edit(pcb,
                          edit_id_str,
                          edit_target_str,
                          edit_operation_str,
                          edit_xml_value,
                          insert_point,
                          insert_where);
    if (res != NO_ERR) {
        log_error("\nAdd edit3 failed for complex test edit (%s)\n",
                  get_error_string(res));
        db_api_free_patch(pcb);
        return;
    }

    /* Step 4: send patch request */
    res = db_api_send_patch(pcb);
    if (res != NO_ERR) {
        log_error("\nSend complex test edit failed (%s)\n",
                  get_error_string(res));
    } else if (LOGDEBUG) {
        log_debug("\nSend complex test edit OK\n");
    }

    /* step 5: free the patch control block */
    db_api_free_patch(pcb);

}  /* send_complex_test_edit */


/********************************************************************
* FUNCTION main
*
* This is an example main function.
*
* RETURNS:
*   0 if NO_ERR
*   status code if error connecting or logging into ncxserver
********************************************************************/
int main (int argc, char **argv)
{
#ifdef MEMORY_DEBUG
    mtrace();
#endif

    /* in case db-lock-test is used */
    (void)time(&start_time);
    last_time = 0;
    locked = FALSE;

    /* 1) setup yumapro messaging service profile */
    status_t res = ycontrol_init(argc, argv,
                                 (const xmlChar *)"subsys1");

    boolean getconfig = false;
    boolean withdef = false;
    boolean with_state = false;
    const char *filespec = NULL;
```

```
    const char *xpath_filter = NULL;
    uint32 cnt = 0;
    uint32 max_count = 0;
    uint32 change_time = 7;
    uint32 st_time = 30;
    boolean entermaint = false;
    boolean exitmaint = false;
    const char *dlevel = NULL;
    uint32 maintbits = 0;
    boolean db_lock_test = FALSE;

    boolean edit_full2 = false;

    if (res == NO_ERR) {
        res = check_cli_parms(argv,
                              &getconfig,
                              &withdef,
                              &with_state,
                              &filespec,
                              &xpath_filter,
                              &max_count,
                              &change_time,
                              &st_time,
                              &entermaint,
                              &exitmaint,
                              &dlevel,
                              &maintbits,
                              &db_lock_test);
        if (res != NO_ERR) {
            print_usage();
        }
    }

    if (getconfig && (filespec==NULL)) {
        res = ERR_NCX_MISSING_PARM;
        print_usage();
    }

    if ((res == NO_ERR) && entermaint && exitmaint) {
        res = ERR_NCX_EXTRA_PARM;
        print_usage();
    }

    /* 2) register services with the control layer */
    if (res == NO_ERR) {
        res = db_api_register_service_ex(db_lock_test);
    }

    /* 3) do 2nd stage init of the control manager (connect to server) */
    if (res == NO_ERR) {
        res = ycontrol_init2();
    }

    boolean done = FALSE;

    /* pick whether the simple edit or the complex edit is sent */
    boolean complex_edit = FALSE;

    /* 4) call ycontrol_check_io periodically from the main program
     * control loop
     */
#ifdef DB_API_APP_DEBUG
    int id = 0;
```

```
#endif  // DB_API_APP_DEBUG

    boolean test_done = FALSE;
    const xmlChar *error_msg = NULL;

    while (!done && res == NO_ERR) {
#ifdef DB_API_APP_DEBUG
        if (LOGDEBUG3) {
            log_debug3("\ndb-api-app: checking ycontrol IO %d", id++);
        }
#endif  // DB_API_APP_DEBUG

        res = ycontrol_check_io();
        if (res != NO_ERR) {
            continue;
        }

        if (ycontrol_shutdown_now()) {
            // YControl has received a <shutdown-event>
            // from the server subsystem is no longer active
            // could ignore or shut down YControl IO loop
            done = TRUE;
        }

        // Using sleep to represent other program work; remove for real
#ifdef DO_SLEEP
        if (!done && (res == NO_ERR)) {
            useconds_t usleep_val = 100000;  // 100K micro-sec == 1/10 sec
            (void)usleep(usleep_val);
        }
#endif  // DO_SLEEP

        if (db_api_service_ready() && !test_done) {
            if (db_lock_test) {
                check_db_lock_test(st_time, change_time, &done);
                continue;
            } else if (getconfig) {
                send_test_getconfig(filespec,
                                    withdef,
                                    with_state,
                                    xpath_filter);
            } else if (complex_edit) {
                /* send the complex test edit */
                send_complex_test_edit();
            } else if (entermaint) {
                send_test_enter_maintmode(maintbits);
            } else if (exitmaint) {
                send_test_exit_maintmode();
            } else if (dlevel) {
                send_test_set_loglevel(dlevel);
            } else if (edit_full2) {
                /*send the test for edit with skip_sil*/
                send_test_edit_full2();
            } else {
                /* send the simple test edit */
                send_test_edit();
            }

            test_done = TRUE;
        } else if (db_api_service_ready() && test_done) {
            /* check the test edit */
            res = db_api_check_edit_ex(&error_msg);
            if (res == NO_ERR) {
```

```
                log_info("\nTest %u succeeded\n", cnt+1);

                if (cnt < max_count) {
                    cnt++;
                    log_debug("\nStart send edit %u", cnt);
                    send_test_edit();
                } else {
                    done = TRUE;
                }
            } else {
                if (res == ERR_NCX_SKIPPED) {
                    res = ERR_NCX_OPERATION_FAILED;
                }

                log_info("\nTest failed with error: %d '%s'\n\n",
                         res, error_msg ? error_msg :
                         (const xmlChar *)get_error_string(res));

                done = TRUE;
            }
        }
    }

    /* 5) cleanup the control layer before exit */
    ycontrol_cleanup();

#ifdef MEMORY_DEBUG
    muntrace();
#endif

    return (int)res;

}  /* main */
```

12.1 DB-API Interface Functions

The DB-API service uses the YControl subsystem similar to the SIL-SA service. The same YControl callback interface is used for handling database access messages.

Refer to **db-api/db_api.h** for details.

Setup

- **db_register_service:** Initialize the DB-API service with the YControl subsystem
- **db_api_service_ready:** Check if the DB-API service is ready to send an edit

Single Edit Patch

- **db_api_send_edit:** Send an edit request to the main server. The SIL and SIL-SA callbacks will not be invoked for this edit. Only the main server datastore will be updated to contain the requested edit. The main server will return an “ok” message if the edit was accepted and an “error” message if the edit was rejected for some reason. The resulting running datastore must pass any YANG validation checks that apply to the changed data somehow.
- **db_api_send_edit_ex**: Same as db_api_send_edit, but with more parameters
- **db_api_send_edit_full**: Same as db_api_send_edit_ex, but with more parameters
- **db_api_send_edit_full2**: Same as db_api_send_full, but with a skip-sil parameter added

Multi-Edit Patch

- **db_api_start_patch**: start a multi-edit patch request
- **db_api_add_edit**: add an edit to the YANG Patch (called 1 – N times)
- **db_api_send_patch**: send the YANG Patch request to the server'
- **db_api_free_patch:** free the memory used by a YANG Patch request

Get Configuration Data

- **db_api_send_getconfig:** send a config request to the server. The running configuration will be retrieved and saved in the specified file.

Get Filtered Configuration or Operational Data

- **db_api_send_getfilter:** send a retrieval request to the server. The running configuration and optionally the operational state will be retrieved and saved in the specified file. An XPath filter can be used to return a subset of the data.

Check Status

- **db_api_check_edit:** check if an edit request or <getconfig> request has finished
- **db_api_check_edit_ex:** Similar to db_api_check_edit but returns error string from server (if any)

Maintenance Mode

- **db_api_send_enter_maintmode:** send a <enter-maintmode> request to the server
- **db_api_send_exit_maintmode:** send a <exit-maintmode> request to the server

DB-LOCK-TEST mode

- **db_api_request_local_db_lock:** send a <db-lock> request to the server
- **db_api_release_local_db_lock:** send a <db-unlock> request to the server

Set Log Level

- **db_api_send_set_loglevel:** send a <set-log-level> request to the server

SubRPC

- **db_api_send_subrpc_request** send a <subrpc-request> message to the server

```
/*********************************************************************
* FUNCTION db_api_register_service
*
*  Register the DB-API service with the YControl layer
*
* RETURNS:
*   status
*********************************************************************/
extern status_t
    db_api_register_service (void);


/*********************************************************************
* FUNCTION db_api_service_ready
*
* RETURNS:
*   TRUE if ready state; FALSE otherwise
*********************************************************************/
extern boolean
    db_api_service_ready (void);


/*********************************************************************
* FUNCTION db_api_send_edit
*
* Create a YANG Patch edit request and send it to the DB-API service
* on the main server.
*
* The content should represent the intended target resource
* as specified in YANG-API (NOT RESTCONF)
* Only the data resource identifier is provided, not the
* API wrapper identifiers (so this can change when RESTCONF is supported)
* Example leaf:
*
*   edit_target == /interfaces/interface/eth0/mtu
*   edit_value == "<mtu>9000</mtu>
*   edit_operation == "merge"
*
* Example list:
*
```

```
*   edit_operation == "create"
*   edit_target == /interfaces/interface/eth0/ipv4
*   edit_value == "<ipv4>
*                   <enabled>true</enabled><
*                   <forwarding>true</forwarding>
*                   <address>204.102.10.4</address>
*                   <prefix-length>24</prefix-length>
*                 </ipv4>"
*
* INPUTS:
*   edit_target == target resource (YANG-API path expression)
*   edit_operation == edit operation (create merge replace delete remove)
*   edit_xml_value == XML payload in string form, whitespace allowed
*                     MAY BE NULL if no value required (delete remove))
* RETURNS:
*   status
*********************************************************************/
extern status_t
    db_api_send_edit (const xmlChar *edit_target,
                      const xmlChar *edit_operation,
                      const xmlChar *edit_xml_value);
```

12.1.1 Filtered Retrieval Example

The **db_api_send_getfilter** function is similar to **db_api_send_getconfig**, but with extended parameters.

```
/*****************************************************************
* FUNCTION db_api_send_getfilter
*
* Create a <getconfig> request and send it to the main server.
* An XPath filter can be sent as a parameter
* INPUTS:
*   filespec == file specification to contain the XML instance
*               document retrieved from the server
*   withdef == TRUE to get with defaults; FALSE to leave out defaults
*   get_config == TRUE for config only; FALSE for get (config + state)
*   xpath_filter == XPath expression (may be nULL)
* RETURNS:
*   status
*****************************************************************/
extern status_t
    db_api_send_getfilter (const xmlChar *filespec,
                           boolean withdef,
                           boolean get_config,
                           const xmlChar *xpath_filter);
```

The **db-api-app** can be used to send a filtered retrieval request.

This example retrieves the RESTCONF monitoring state information:

```
> db-api-app --getconfig --withdef --with-state \
    --xpath-filter=/restconf-state --filespec=test.xml
```

test.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="http://yumaworks.com/ns/yumaworks-db-api">
  <restconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
    <capabilities>
      <capability>urn:ietf:params:restconf:capability:depth:1.0</capability>
      <capability>urn:ietf:params:restconf:capability:with-defaults:1.0</capability>
      <capability>urn:ietf:params:restconf:capability:defaults:1.0?basic-mode=explicit</
capability>
      <capability>urn:ietf:params:restconf:capability:fields:1.0</capability>
      <capability>urn:ietf:params:restconf:capability:replay:1.0</capability>
      <capability>urn:ietf:params:restconf:capability:filter:1.0</capability>
      <capability>urn:ietf:params:restconf:capability:yang-patch:1.0</capability>
    </capabilities>
    <streams>
      <stream>
        <access>
          <encoding>xml</encoding>
```

```
        <location>http://localhost/restconf/stream</location>
      </access>
      <description>default RESTCONF event stream</description>
      <name>RESTCONF</name>
      <replay-log-creation-time>2018-06-05T16:17:40Z</replay-log-creation-time>
      <replay-support>true</replay-support>
    </stream>
  </streams>
 </restconf-state>
</config>
```

12.1.2 Subsystem RPC Request Example

The <subrpc-request> message is used by a subsystem to send an RPC Operation Request to the server.

The <subrpc-response> message is used by the server to send an RPC reply message to the subsystem that sent the subrpc-request message.

The **db_api_send_subrpc_request** function is used for this operation:

```
/*************************************************************
* FUNCTION db_api_send_subrpc_request
*
* Create a <subrpc-request> request and send it to the main server.
* INPUTS:
*   user_id == user name to run RPC on server; NULL == system user
*   rpc_modname == module name containing the RPC or NULL to scan
*                  all modules for the first match of rpc_name
*   rpc_name == name of the RPC (must be present)
*   in_filespec == file specification to contain the XML instance
*              document for the rpc-method element (if needed)
*   out_filespec == file specification to contain the XML instance
*              document retrieved from the server (must be present)
* OUTPUTS:
*   file named by out_filespec created if operation succeeded.
* RETURNS:
*   status
*************************************************************/
extern status_t
    db_api_send_subrpc_request (const xmlChar *user_id,
                                const xmlChar *rpc_modname,
                                const xmlChar *rpc_name,
                                const xmlChar *in_filespec,
                                const xmlChar *out_filespec);
```

The rpc1.yang module is used for this example:

```
module rpc1 {
  namespace "http://www.yumaworks.com/ns/rpc1";
  prefix r1;
  revision "2020-02-25";

  rpc rpc1 {
    input {
```

```
      leaf A {
        type string;
      }
      leaf B {
        type int32;
      }
    }
    output {
      leaf C {
        type string;
      }
      leaf D {
        type int32;
      }
    }
   }

} // module rpc1
```

The input file rpc1.xml:

```
<rpc1 xmlns="http://www.yumaworks.com/ns/rpc1">
  <A>test</A>
  <B>11</B>
 </rpc1>
```

The code to send the RPC request to the server:

```
#include “dp_api.h”

….

const xmlChar *user_id = (const xmlChar *)”admin1”;
const xmlChar *rpc_modname = (const xmlChar *)”rpc1”;
const xmlChar *rpc_name = (const xmlChar *)”rpc1”;
const xmlChar *in_filename = (const xmlChar *)”rpc1.xml”;
const xmlChar *out_filename = (const xmlChar *)”rpc1-out.xml”;

status_t res =
    db_api_send_subrpc_request(user_id,
                               rpc_modname,
                               rpc_name,
                               in_filename,
                               out_filename);
```

The output file rpc1-out.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="http://www.yumaworks.com/ns/rpc1">
  <C>test1</C>
```

```
  <D>14</D>
</rpc-reply>
```

12.1.3 Subsystem Action Request Example

A YANG Action Request is just a special form of the RPC operation. The same format is used here as in the NETCONF protocol for the YANG action RPC.

The ex-action.yang module is used for the example:

```
module ex-action {
  yang-version 1.1;
  namespace "http://netconfcentral.org/ns/ex-action";
  prefix exa;
  import ietf-yang-types { prefix yang; }
  revision 2020-03-06;

  list server {
    key name;
    leaf name {
      type string;
      description "Server name";
    }
    action reset {
      input {
        leaf reset-msg {
          type string;
          description "Log message to print before server reset";
        }
      }
      output {
        leaf reset-finished-at {
          type yang:date-and-time;
          description "Time the reset was done on the server";
        }
      }
    }
  }
}
```

The act1.xml input file:

```
<action xmlns="urn:ietf:params:xml:ns:yang:1">
 <server xmlns="http://netconfcentral.org/ns/ex-action">
  <name>test1</name>
  <reset>
   <reset-msg>this is a reset message</reset-msg>
  </reset>
 </server>
```

```
 </action>
```

The code to send the action request to the server:

```
#include “dp_api.h”

….

const xmlChar *user_id = (const xmlChar *)”admin1”;
const xmlChar *rpc_modname = (const xmlChar *)”ex-action”;
const xmlChar *rpc_name = (const xmlChar *)”reset”;
const xmlChar *in_filename = (const xmlChar *)”act1.xml”;
const xmlChar *out_filename = (const xmlChar *)”act1-out.xml”;

status_t res =
    db_api_send_subrpc_request(user_id,
                               rpc_modname,
                               rpc_name,
                               in_filename,
                               out_filename);
```

The output file act1-out.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="http://netconfcentral.org/ns/ex-action">
  <reset-finished-at>2020-03-11T19:50:30Z</reset-finished-at>
</rpc-reply>
```

12.2 DB-API Messages

The DB-API messages are defined in the **yumaworks-db-api.yang** module.

The only message defined at this time is “edit-request”. This is sent as a “subsys-request” by the subsystem to the main server. The server will send an “ok” or “error” reply as a ”server-response” message.

```
module yumaworks-db-api {

    namespace "http://yumaworks.com/ns/yumaworks-db-api";

    prefix "ydb";

    import ietf-yang-types { prefix yang; }
    import ietf-yang-patch { prefix yp; }
    import yumaworks-ycontrol { prefix yctl; }
    import yuma-ncx { prefix ncx; }
    import yuma-types { prefix yt; }

    organization "YumaWorks, Inc.";

    contact
        "Support <support at yumaworks.com>";

    description
       "YumaPro Database API Sub-Agent message definitions.

     Copyright (c) 2014 - 2020, YumaWorks, Inc. All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the BSD 3-Clause License
     http://opensource.org/licenses/BSD-3-Clause";

    revision 2020-02-19 {
        description
          "Add subrpc-request and subrpc-response messages.";
    }

    revision 2019-01-27 {
        description
          "Add allowed parameter to maintenance mode.
           Add db-lock messages";
    }

    revision 2018-11-13 {
        description
          "Add maintenance mode and set-loglevel support";
    }

    revision 2018-06-13 {
        description
          "Add skip-sil parameter to edit-request message";
    }

    revision 2018-05-31 {
        description
          "Add parameters to getconfig message";
```

```
  }

  revision 2017-10-30 {
      description
        "Add getconfig message";
  }

  revision 2017-09-19 {
      description
        "Fix broken augment statement for payload";
  }

  revision 2016-08-29 {
      description
        "Update yp-ha-mode event";
  }

  revision 2016-07-31 {
      description
        "Add new events to support YP-HA mode changes.";
  }

  revision 2015-03-29 {
      description
        "Add edit-type parameter to edit-request.";
  }

  revision 2014-11-18 {
      description
        "Initial version for datastore access.";
  }

  grouping lock-id {
    leaf lock-id {
      type string {
        length "1 .. 64";
      }
      mandatory true;
      description
        "The lock identifier sent by the server in the
         db-lock request. Used to prevent late replies
         from being interpreted as a reply to a new request.";
    }
  }

  augment "/yctl:ycontrol/yctl:message-payload/yctl:payload/yctl:payload" {
    container db-api {

      choice message-type {
        mandatory true;

        leaf register-request {
          type empty;
          description
           "Message type: subsys-request;
            Purpose: register the DB-API subsystem
            Expected Response Message: ok or error";
        }

        container edit-request {
          description
           "Message type: subsys-request;
            Purpose: Ask the main server to accept an edit request
```

```
              to be added to the running configuration, and saved to
              NV-storage unless the :startup capability is supported.

              Expected Response Message: ok or error";

            leaf target {
              type string;  // target-path-string
              mandatory true;
              description
               "Identifies the target resource for the edit
                operation.";
            }

            leaf edit-type {
              type enumeration {
                enum user {
                  description "A user edit with access control";
                }
                enum system {
                  description "A system edit without access control";
                }
              }
              default user;
              description
               "Indicates whether this is a user edit or system edit.
                System edits will bypass all access control enforcement,
                including the ncx:user-write extension.";
            }

            leaf skip-sil {
              type boolean;
              default true;
              description
               "Skip the SIL and SIL-SA callbacks for this transaction.
                This is the normal mode since the DB-API edit is
                generated by the system, so the edits do not need
                to be applied again by the server.";
            }

            uses yp:yang-patch;

          }  // container edit-request

          container yp-ha-mode {
            description
             "Message type: subsys-event;
              Purpose: send mode change event to the server
              Expected Response Message: none";

            choice action {
              mandatory true;
              leaf go-active {
                type empty;
                description
                  "Become the YP-HA active server.
                   All normal management operations are supported
                   in this mode.";
              }
              container go-standby {
                description
                  "Become a YP-HA standby server, and try to connect
                   to the active server 'new-active'. Only the superuser
                   can use management sessions in this mode.";
```

```
        leaf new-active {
          type string;
          mandatory true;
          description
            "Server name of the active server to use";
        }
      }
      leaf go-none {
        type empty;
        description
          "Leave current YP-HA role and wait new role.
           Only the superuser can use management sessions
           in this mode.";
      }

    }
  }  // container yp-ha-mode

  container getconfig {
    description
     "Message type: subsys-request;
      Purpose: Ask the main server to send the running
      configuration contents

      Expected Response Message: config";

      leaf withdef {
        type boolean;
        default false;
        description
          "Include defaults (according to the server
           --default-style CLI parameter) if 'true'.
           Do not include defaults if 'false'.";
      }

      leaf with-state {
        type boolean;
        default false;
        description
          "Include operational data (like <get> operation)";
      }

      leaf xpath-filter {
        type yang:xpath1.0;
        description
          "XPath filter to use for this retrieval operation";
      }

  }  // container getconfig

  container config {
    ncx:root;
    description
     "Message type: server-response;
      Purpose: Provide the contents of the running
      configuration contents

      Expected Response Message: none";

  }  // container config
```

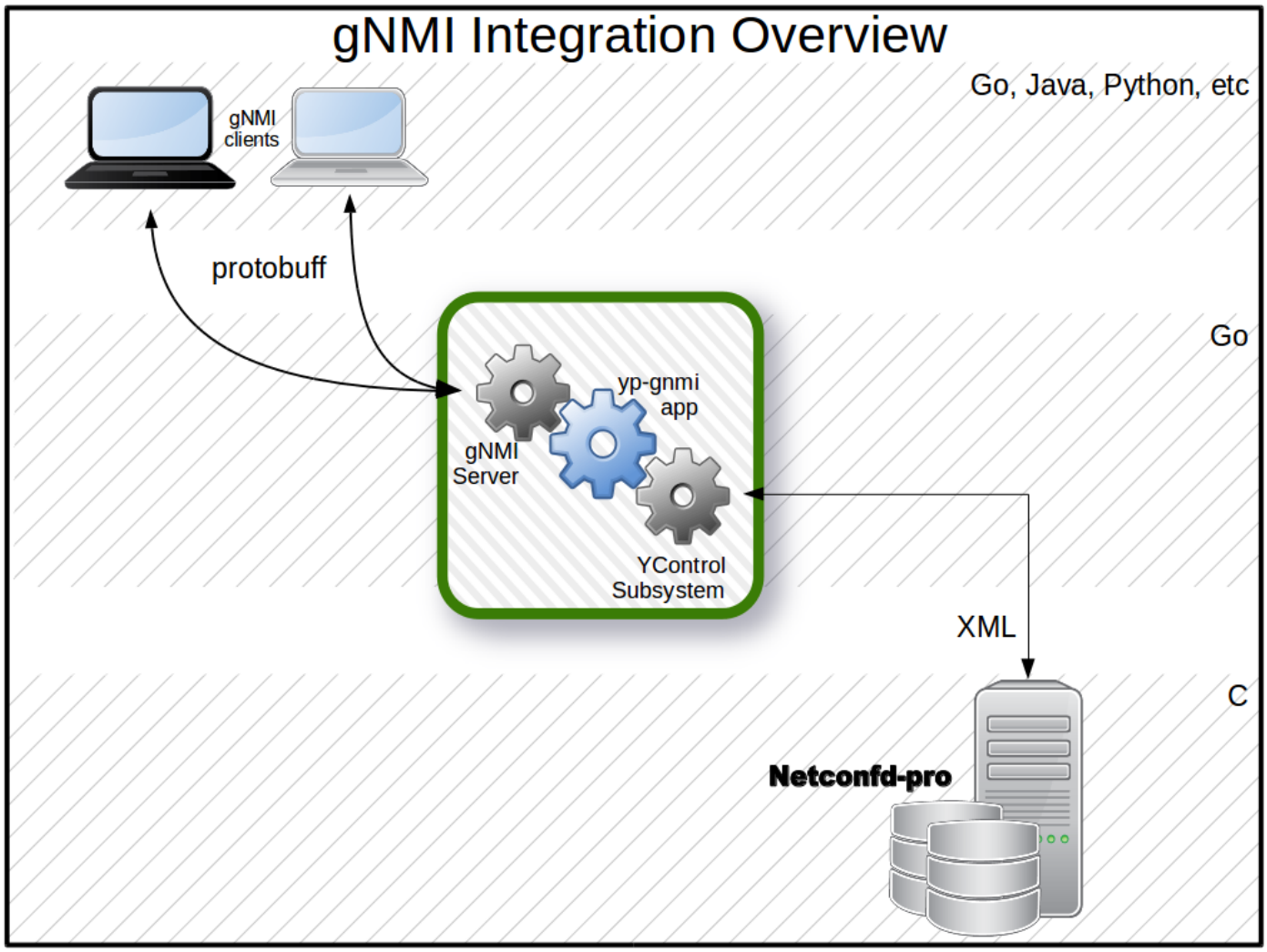
```
        container enter-maintmode {
          description
           "Message type: subsys-request.
            Purpose: Enter maintenance mode.
            Expected Response Message: server-response
            The server will send ok or error.";

          leaf allowed {
            type bits {
              bit read {
                position 0;
                description
                  "Allow client sessions for operations that
                   read datastores or operational data.";
              }
              bit operation {
                position 1;
                description
                  "Allow client sessions for general operations
                   that do not access any datastores.";
              }
            }
            default "";
            description
              "The client activity that is allowed during
               maintenance mode. By default client sessions
               are disabled during maintenance mode, and any
               existing sessions will get 'access-denied' errors
               for all operations started after maintenance mode
               is started.";
          }
        }  // container enter-maintmode

        container exit-maintmode {
          description
           "Message type: subsys-request.
            Purpose: Exit maintenance mode.
            Expected Response Message: server-response
            The server will send ok or error.";

        }  // container exit-maintmode

        container set-log-level {
          description
           "Message type: subsys-request.
            Purpose: Set the log-level parameter.
            Expected Response Message: server-response
            The server will send ok or error.";

          leaf log-level {
            type yt:NcDebugType;
            mandatory true;
            description "The new log level to set";
          }

        }  // container set-log-level

        container db-lock-init {
          description
           "Message type: subsys-request.
            Purpose: Establish the subsystem as the DB-Edit-Lock
            controller. This is done once after the db-api service
            is registered.
```

```
            Expected Response Message: server-response
            The server will send ok or error.";
        }

        container db-lock {
          description
           "Message type: server-request.
            Purpose: Request the system config write lock.
            Expected Response Message: subsys-response
            The subsystem will send db-lock-response or error.";
          uses lock-id;
        }

        container db-lock-response {
          description
           "Message type: subsys-response.
            Purpose: Response for the system config write lock.
            Expected Response Message: none";
          uses lock-id;
        }

        container db-unlock {
          description
           "Message type: server-event.
            Purpose: Release the system config write lock.
            Expected Response Message: none";
          uses lock-id;
        }

        container subrpc-request {
          description
           "Message type: subsys-request;
            Purpose: Start an RPC transaction to the main server.

            This message requests that a new remote procedure
            call be validated and invoked on the server and
            possibly one or more SIL-SA subsystems.

            If not valid or if the operation cannot be performed,
            the server must return an error.

            Expected Response Message: subrpc-response";

          leaf user-id {
            type string;
            description
              "Identifies the user that should be used for
               access control and logging purposes.
               If not present then the system user will be used
               and all access control will be skipped.";
          }

          leaf rpc-module {
            type yt:NcxName;
            description
              "Identifies the module name of the RPC definition.
               If not present then the server will use the first
               RPC method with the same name as 'rpc-name'.";
          }

          leaf rpc-name {
            type yt:NcxName;
            mandatory true;
```

```
            description
              "Identifies the name of the RPC definition.";
          }

          anyxml rpc-method {
            description
              "Contains the RPC method element and all child nodes
               representing the input parameters. This is structured
               exactly the same as the contents of an <rpc> element
               from RFC 6241. If not present then an empty method
               element will be constructed using 'rpc-module' and
               'rpc-name' values.";
          }
        }

        container subrpc-response {
          description
           "Message type: server-response
            Purpose: Return <rpc-reply> message
            Expected Response Message: none";

          leaf rpc-module {
            type yt:NcxName;
            mandatory true;
            description
              "Identifies the module name of the RPC definition.";
          }

          leaf rpc-name {
            type yt:NcxName;
            mandatory true;
            description
              "Identifies the name of the RPC definition.";
          }

          anyxml reply {
             mandatory true;
             description
               "Contains the RPC reply message for the subrpc-request.";
          }
        }

      } // choice message-type
    } // container db-api
  } // augment
}
```

13 YumaPro gNMI Subsystem

The following diagram illustrates key components of the gNMI and **netconfd-pro** integration.



As illustrated above gNMI clients can be written in any languages that are supported for gNMI clients. The client part is out of the scope of this document and the current gNMI protocol integration does not include client part. The clients communicate to the target (**netconfd-pro**) with help of the **ypgnmi-app** application and send gRPC request to the application.

The **ypgnmi-app** application is written in GO language and talk to the **netconfd-pro** server via socket with XML encoding and acts as a YControl subsystem.

The main role of **ypgnmi-app** application is to translate clients requests to XML and contact **netconfd-pro** server and send the replies back to the gNMI clients.

The **ypgnmi-app** application is a YControl subsystem that communicates to the **netconfd-pro** server and also it is a gNMI server that communicates to the gNMI clients. Also, the **ypgnmi-app** application is responsible for encoding conversion between gNMI gRCP to XML that are sent to the **netconfd-pro** server and vise versa.

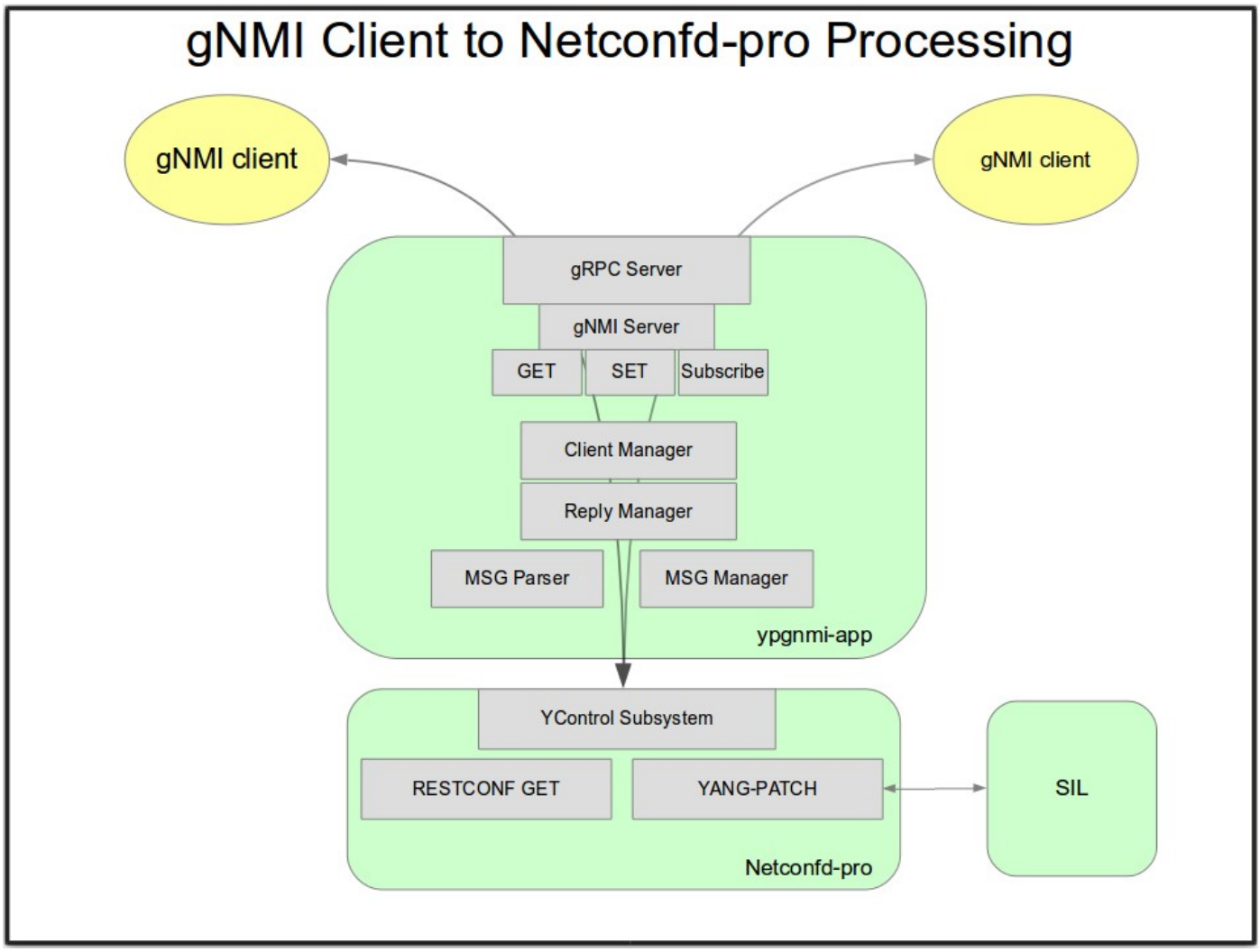
The messages processing between gNMI client to the **netconfd-pro** server can be split into the following components:

- gNMI clients to the **ypgnmi-app** application processing - includes message parsing and conversion of the messages to the XML message

- The **ypgnmi-app** application to the **netconfd-pro** server messages processing. YControl messages exchange encoded in XML
- The **netconfd-pro** server internal processing – includes subsystem registration, subsystem messages handling and parsing, conversion to RESTCONF RCB for the request processing.
  - YANG-PATCH in case of gNMI SetRequest
  - RESTCONF GET in case of gNMI GetRequest and Subscribe with mode ONCE

13.1 gNMI Client to Netconfd-pro Processing

The following diagram illustrates the messages processing path from gNMI clients to the **netconfd-pro** server and to the database for configuration and retrieval.



13.1.1 Ypgnmi-app Processing

The **ypgnmi-app** implements multiple goroutines to manage the replies and the clients. The following goroutines are implemented in the **ypgnmi-app:**

- **Client Manager goroutine**. Responsible for all the gNMI clients connections. After the client contacts the gNMI server, this goroutine register the client, store its information and run all the authentication processing. It verifies the certificates and triggers the connection to be shutdown.
- **Reply Manager goroutine.** This manager is responsible for any already parsed messages from the **netconfd-pro** server or gNMI client, it stores any not processed messages that are ready to be processed. It is responsible to find the corresponding clients that wait for the response and triggers the response procedure.

- **Message Parser goroutine.** It is responsible for all the messages parsing. It parses messages from clients encoded in the protobuf and convert them into XML encoded messages to be send to the **netconfd-pro** server and vice versa.
- **Message Manager goroutine.** This manager is responsible for storing any ready to be processed messages that are going to the **netconfd-pro** server and that are coming back from the server.

All of this managers are goroutines. They run in parallel and asynchronously.

The core of the **ypgnmi-app** is the gNMI server that is build on top of gRPC server. It contains gNMI server code that responsible for the:

- gNMI client to the **netconfd-pro** server communication
- Handle GET/SET/Subscribe/Capability callbacks. Send/get messages from the **netconfd-pro** server and response to the clients.
- Register gNMI server
- Register gRPC server for the protobuf message handling
- Run main Serve loop that handles all the client/server communication

The **ypgnmi-app** application implements the HTTP/2 server over TLS in Go standard package **“golang.org/x/net/http2”**. This package provides authentication, connection, listen, etc handlers that fully fulfill all the requirements for the **ypgnmi-app** application.

The **ypgnmi-app** application has the following startup steps for the gNMI server component:

- Initialize all the prerequisites and parse all the CLI parameters
- Based on the **netconfd-pro** server modules capability create gNMI target
- Open TCP socket to listen for clients requests
- Serve any incoming gNMI messages from gNMI clients and re-transmit them to the **netconfd-pro** server if needed with help of all the goroutine managers.

On the other side the **ypgnmi-app** acts as a YControl subsystem (similar to **db-api-app**), however, it does not terminate after one edit or get request. Instead it continuously listens to the **netconfd-pro** server and keeps the AF_LOCAL or TCP socket open to continue communication whenever it's needed. The communication is terminated only if the **ypgnmi-app** application is terminated or the **netconfd-pro** server terminates.

All the message definitions described in the **yumaworks-yp-gnmi.yang** YANG module and similar to the original gNMI .proto message definitions which makes the conversion easier and faster.

The **ypgnmi-app** application has the following startup steps to initialize connection with the **netconfd-pro** server:

- Initialize all the prerequisites and parse all the CLI parameters
- Open socket and send <ncx-connect> request to the server with
  - transport = netconf-aflocal
  - protocol = yp-gnmi
- Register yp-gnmi service
  - Send <register-request> to the server
  - Register **yp-gnmi** subsystem and initialize all corresponding code in the **netconfd-pro** server to be ready to handle **yp-gnmi** application requests
- Create data model structure that will be used for gNMI client/server communication
  - Send <config-request> to the server
  - Feed the gNMI ModelData with all supported modules

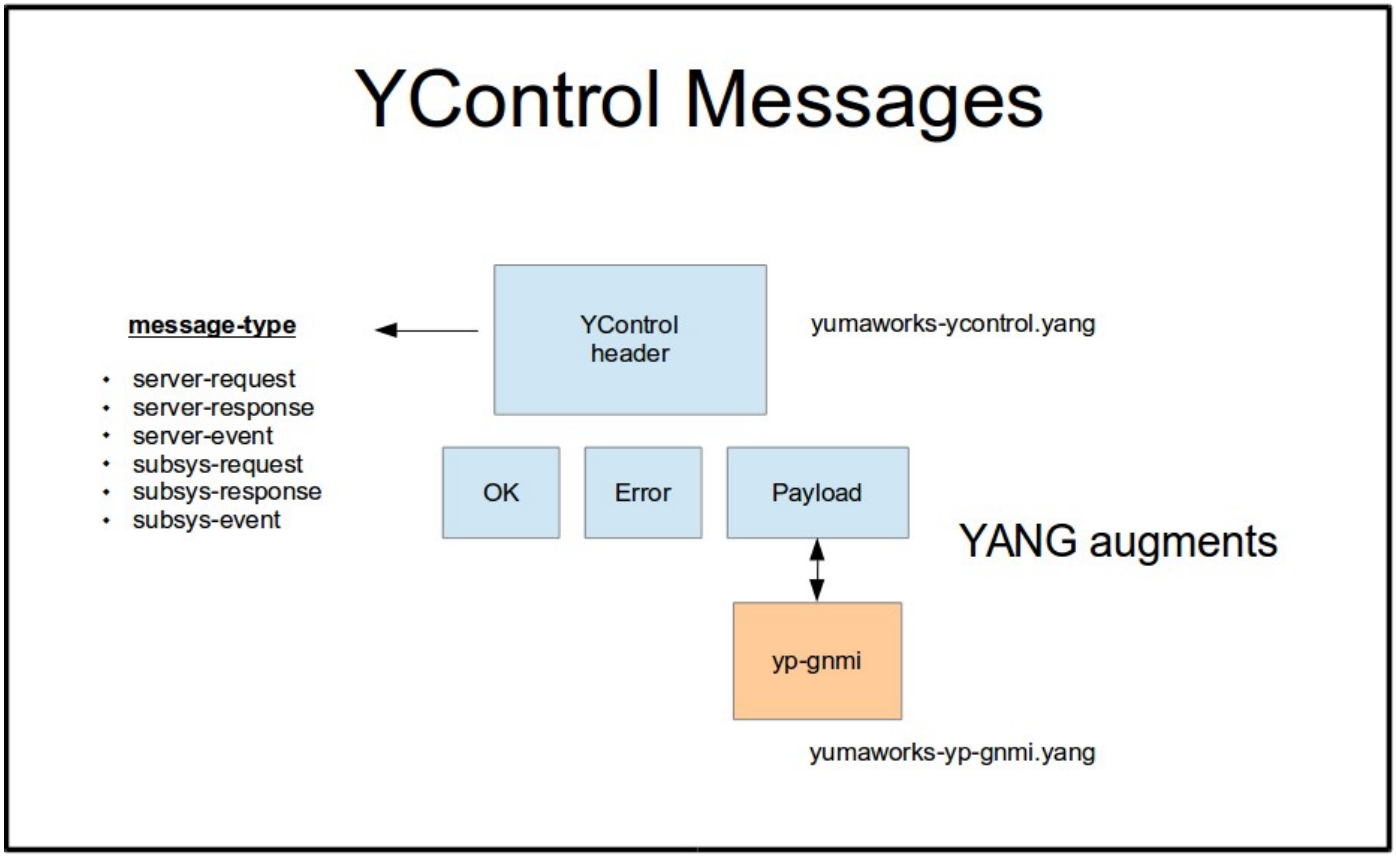
- Keep listening socket until terminated

13.2 Ypgnmi-app Message Format

The **ypgnmi-app** application uses several messages to interact with the **netconfd-pro** server.

These messages are defined in the yumaworks-yp-gnmi YANG module. The **ypgnmi-app** payload is defined as a YANG container that augments the YControl “message-payload” container.

The following diagram illustrates the YControl messages overview:



13.2.1 Ypgnmi-app Registration Message Flow

During the startup phase the server will initialize the yp-gnmi subsystem callback functions and handlers (similar way as for db-api module does).

The connection with the server is getting started with <ncx-connect> message that adds the YControl subsystem with the "yp-gnmi" subsystem ID to the server (**agt_connect** module).

YControl protocol connection parameters:

- transport: **netconf-aflocal**
- protocol: **yp-gnmi**
- <port-num> not used

Additional parameters:

- subsys-id: **yp-gnmi**

The Registration message flow looks as follows:

- **Ypgnmi-app to Server:** *config-request:*
  The service requests the **netconfd-pro** server its configuration in 1 message.
- **Server to Ypgnmi-app:** *config-response:*
  The server responds to the config request with a list of modules that need to be loaded to gNMI server DataModel
- **Ypgnmi-app to Server:** *register-request:*
  The **ypgnmi-app** service registers callbacks supported by the subsystem.
- **Server to Ypgnmi-app:** *ok:*
  The server responds to the register request with an <ok> or an <error> message

The module "yumaworks-yp-gnmi.yang" defines all the messages

13.2.2 Yumaworks-yp-gnmi YANG Module

```
module yumaworks-yp-gnmi {
    yang-version 1.1;

    namespace "http://yumaworks.com/ns/yumaworks-yp-gnmi";

    prefix "ypgnmi";

    import yumaworks-ycontrol { prefix yctl; }
    import yuma-types { prefix yt; }

    organization "YumaWorks, Inc.";

    contact
        "Support <support at yumaworks.com>";

    description
      "Package gNMI defines a service specification for the gRPC
       Network Management Interface. This interface is defined to
       be a standard interface via which a network management system
       ('client') can subscribe to state values, retrieve snapshots of
       state information, and manipulate the state of a data tree
       supported by a device ('target').

       This document references the gNMI Specification which can be
       found at:
       http://github.com/openconfig/reference/blob/master/rpc/gnmi

       Licensed under the Apache License, Version 2.0 (the 'License');
       you may not use this file except in compliance with the License.
       You may obtain a copy of the License at:
       http://www.apache.org/licenses/LICENSE-2.0


       YumaPro gNMI message definitions.
       Copyright (c) 2014 - 2019, YumaWorks, Inc. All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the BSD 3-Clause License
       http://opensource.org/licenses/BSD-3-Clause
       ";


    revision 2018-02-05 {
        description
          "Initial version for datastore access.";
    }

    typedef supported-encoding {
      type enumeration {
        enum json_val {
          description "A JSON encoded string";
        }
        enum bytes_val {
          description "An arbitrary sequence of bytes";
        }
        enum any_val {
```

```
        description "A Protobuf encoded message using protobuf.any";
      }
      enum ascii_val {
        description
          "An ASCII encoded string representing text
           formatted according to a target-defined convention";
      }
      enum json_ietf_val {
        description
          "A JSON encoded string using JSON encoding compatible
           with [RFC 7951]";
      }
    }
  }

  // TBD if we ever need this filed.
  grouping extension-field {
    description
     "The Extension message is defined within the gnmi_ext.proto
      specification. It is carried as a repeated field within each
      of the top-level request and response gNMI messages.

      The Extension message consists of a single oneof which may contain:
       - A well-known extension. Each well known extension defined
         in the gnmi_ext.proto file will be added to the oneof and
         assigned a unique field tag.
       - A registered extension, expressed as a RegisteredExtension message.

      The subfields of this message are:
       - An enumerated id field used to store the unique ID assigned
         to the registered extension.
       - A bytes field which stores the binary-marshalled protobuf
         for the extension.";

    leaf-list extension {
      type union {
        type enumeration {
          enum EID_UNSET;
          enum EID_EXPERIMENTAL {
            description
              "An experimental extension that may be used during
               prototyping of a new extension.";
          }
        }

        type binary;
      }
    }
  }

  grouping model-data {
    description
     "The ModelData message describes a specific model that is supported
      by the target and used by the client. The fields of the ModelData
      message identify a data model registered in a model catalog,
      as described in [draft-openconfig-netmod-model-catalog]
      (the schema of the catalog itself - expressed in YANG - is described
      in [openconfig-module-catalog.yang]).
      Each model specified by a ModelData message may refer to a specific
      schema module, a bundle of modules, or an augmentation or deviation,
      as described by the catalog entry.

      Each ModelData message contains the following fields:
```

```
       - name - name of the model expressed as a string.
       - organization - the organization publishing the model, expressed
         as a string.
       - version - the supported (or requested) version of the model,
         expressed as a string which represents the semantic version of
         the catalog entry.

       The combination of name, organization, and version uniquely identifies
       an entry in the model catalog.";

     list supported-modules {
       key "name";

       description
         "A set of ModelData messages describing each of the
          models supported by the target";

       leaf name {
         type string;
         description
           "Name of the model expressed as a string.";
       }

       leaf organization {
         mandatory true;
         type string;
         description
           "The organization publishing the model, expressed
            as a string.";
       }

       leaf version {
         mandatory true;
         type string;
         description
           "The supported (or requested) version of the model,
            expressed as a string which represents the semantic version
            of the catalog entry.";
       }
     }
   }

   grouping update-list {
     description
      "A re-usable Update message is utilised to indicate changes to
       paths where a new value is required. The Update message contains
       two fields";

     list update {
       key "path";
       description
         "A list of update messages that indicate changes in the underlying
          data of the target. Both modification and creation of data is
          expressed through the update message.";

       leaf path {
         type string;
         description
           "Paths are represented according to gNMI Path Conventions,
            which defines a structured format for path elements, and any
            associated key values.";
       }
```

```
      leaf typed-value {
        // the actual type will be parsed and set in the
        // ypgnmi-app. This is basically plain JSON string.
        type string;

        description
          "The value of a data node (or subtree) is encoded in a
           TypedValue message as a oneof field to allow selection of
           the data type by setting exactly one of the member fields.

           TypedValue is used to encode a value being sent between the
           client and target (originated by either entity).

           message TypedValue {
            One of the fields within the val oneof is populated with the
            value of the update. The type of the value being included in
            the Update determines which field should be populated. In
            the case that the encoding is a particular form of the base
            protobuf type, a specific field is used to store the
            value (E.g., json_val).

            oneof value {
              string string_val = 1;            // String value.
              int64 int_val = 2;                // Integer value.
              uint64 uint_val = 3;              // Unsigned integer value.
              bool bool_val = 4;                // Bool value.
              // Arbitrary byte sequence value.
              bytes bytes_val = 5;
              float float_val = 6;              // Floating point value.
              Decimal64 decimal_val = 7;        // Decimal64 encoded value.
              // Mixed type scalar array value.
              ScalarArray leaflist_val = 8;
              // protobuf.Any encoded bytes.
              google.protobuf.Any any_val = 9;
              bytes json_val = 10;              // JSON-encoded text.
              // JSON-encoded text per RFC7951.
              bytes json_ietf_val = 11;
              string ascii_val = 12;            // Arbitrary ASCII text.
            }
           }
          ";
      }

      leaf duplicates {
        type uint32;
        description
          "A counter value that indicates the number of coalesced
           duplicates. If a client is unable to keep up with the server,
           coalescion can occur on a per update (E.g, per path) basis
           such that the server can discard previous values for a given
           update and return only the latest. In this case the server
           SHOULD increment a count associated with the update such that
           a client can detect that transitions in the state of the path
           have occurred, but were suppressed due to its inability to
           keep up.";
      }
    }
  }

  grouping notifications {
    description
     "When a target wishes to communicate data relating to the state of
      its internal database to an interested client, it does so via means
```

```
      of a common Notification message. Notification messages are reused
      in other higher-layer messages for various purposes.

      The creator of a Notification message MUST include the timestamp
      field. All other fields are optional.";

    list notification {
      key "timestamp";

      leaf timestamp {
        type string;
        description
          "The time at which the data was collected by the device
           from the underlying source, or the time that the target generated
           the Notification message (in the case that the data does not
           reflect an underlying data source).";
      }

      leaf prefix {
        type string;
        description
          "A prefix which is applied to all path fields included in the
           Notification message. The paths expressed within the message are
           formed by the concatenation of prefix + path.
           The prefix always precedes the path elements.";
      }

      leaf alias {
        type string;
        description
          "A string providing an alias for the prefix specified within
           the notification message.";
      }

      uses update-list;

      leaf-list delete {
        type string;
        description
          "A list of paths that indicate the deletion of data nodes
           on the target.";
      }
    }
  }

  grouping config-parms {
    leaf-list module {
      type yt:NcModuleSpec;
        description "Module libraries to use to generate gNMI datastore.
          Generated datastore package contains definitions of structs
          which represent a YANG schema.

          The datastore representation written in go, this
          datastore let us convert/validate/parse YANG to XML/JSON
          to ProtoBuff";
    }
  }


  augment "/yctl:ycontrol/yctl:message-payload/yctl:payload/yctl:payload" {
    container yp-gnmi {

      choice message-type {
```

```
        mandatory true;

        case config-request-case {
          leaf config-request {
            type empty;
            description
             "Message type: subsys-request;
              Purpose: register the service with the server
              and request the service configuration from server.
              Expected Response Message: config-response";
          }
        }

        case register-request-case {
          leaf register-request {
            type empty;
            description
             "Message type: subsys-request;
              Purpose: register the YP-gNMI subsystem
              Expected Response Message: ok or error";
          }
        }

        container config-response {
          description
           "Message type: server-reply;
            Purpose: server will send this element containing the
            requested sub-system configuration.
            Expected Response Message: none";

          uses config-parms;
        }


        // Not sure if this is needed anymore ??
        // After the datastore package is generated it can be used
        // for DataModel generation
        container capability-request {
          description
           "Message type: subsys-request;
            Purpose: The CapabilityRequest message is sent to the server
            to request capability information from the server.
            The CapabilityRequest message carries a single repeated
            extension field.

            Expected Response Message: capability-response";

            uses extension-field;
        }

        container capability-response {
          description
           "Message type: server-response;
            Purpose: message that includes its YP-gNMI service version,
            the versioned data models it supports, and the supported
            data encodings.
            This information is used in subsequent RPC messages from
            the client to indicate the set of models that the client will
            use (for Get, Subscribe, etc), and the encoding to be used
            for the data.

            Expected Response Message: none";
```

```
      // getting handled by config-request (sil-sa style request)
      container supported-modules {
        uses model-data;
      }

      // Can be hard-wired to only JSON (So not needed)
      leaf supported-encodings {
        type supported-encoding;

        default json_val;
        description
         "An enumeration field describing the data encodings supported
          by the target";
      }

      // hard-wired value (So not needed)
      leaf gNMI-version {
        type string;
        description
         "The semantic version of the gNMI service supported by the
          target, specified as a string.";
      }

      uses extension-field;
    } // capability-response

    container get-request {
      description
       "Message type: subsys-request;
        Purpose: Ask the main server to send the running
        configuration contents

        Expected Response Message: get-response";

      // May be ommitted and the next 'path' leaf-list
      // can be used to serialized get request targets
      leaf prefix {
        type string;
        description
          "Paths are represented according to gNMI Path Conventions,
           which defines a structured format for path elements,
           and any associated key values.

           In a number of messages, a prefix can be specified to
           reduce the lengths of path fields within the message.
           In this case, a prefix field is specified within a
           message - comprising of a valid path.
           In the case that a prefix is specified, the absolute
           path is comprised of the concatenation of the list of
           path elements representing the prefix and the list of
           path elements in the path field.";
      }

      leaf-list path {
        type string;
        description
          "A set of paths for which the client is requesting a
           data snapshot from the target.
           The path specified MAY utilize wildcards.
           In the case that the path specified is not valid,
           the target MUST return an RPC response indicating
           an error code of InvalidArgument and SHOULD provide
           information about the invalid path in the error message
```

```
            or details.";
        }

        leaf type {
          type enumeration {
            enum config;
            enum state;
            enum oper;
            enum all;
          }

          default all;
          description
            "The type of data that is requested from the target.
             The valid values for type are described below.

             The types of data currently defined are:
              - CONFIG - specified to be data that the target considers
                to be read/write.
                If the data schema is described in YANG, this corresponds
                to the 'config true' set of leaves on the target.
              - STATE - specified to be the read-only data on the target.
                If the data schema is described in YANG, STATE data is
                the 'config false' set of leaves on the target.
              - OPERATIONAL - specified to be the read-only data on the
                target that is related to software processes operating
                on the device, or external interactions of the device.

              If the type field is not specified, the target MUST return
              CONFIG, STATE and OPERATIONAL data fields in the tree
              resulting from the client's query.";
        }

        leaf encoding {
          type supported-encoding;

          default json_val;
          description
            "The encoding that the target should utilise to serialise
             the subtree of the data tree requested.

             If the Capabilities RPC has been utilised, the client SHOULD
             use an encoding advertised as supported by the target.

             If the encoding is not specified, JSON MUST be used.
             If the target does not support the specified encoding,
             the target MUST return an error of Unimplemented.
             The error message MUST indicate that the specified encoding
             is unsupported.";
        }

        container use-models {
          description
            "A set of ModelData messages indicating the schema definition
             modules that define the data elements that should be returned
             in response to the Get RPC call.";

          uses model-data;
        }

        uses extension-field;
      }  // container getconfig
```

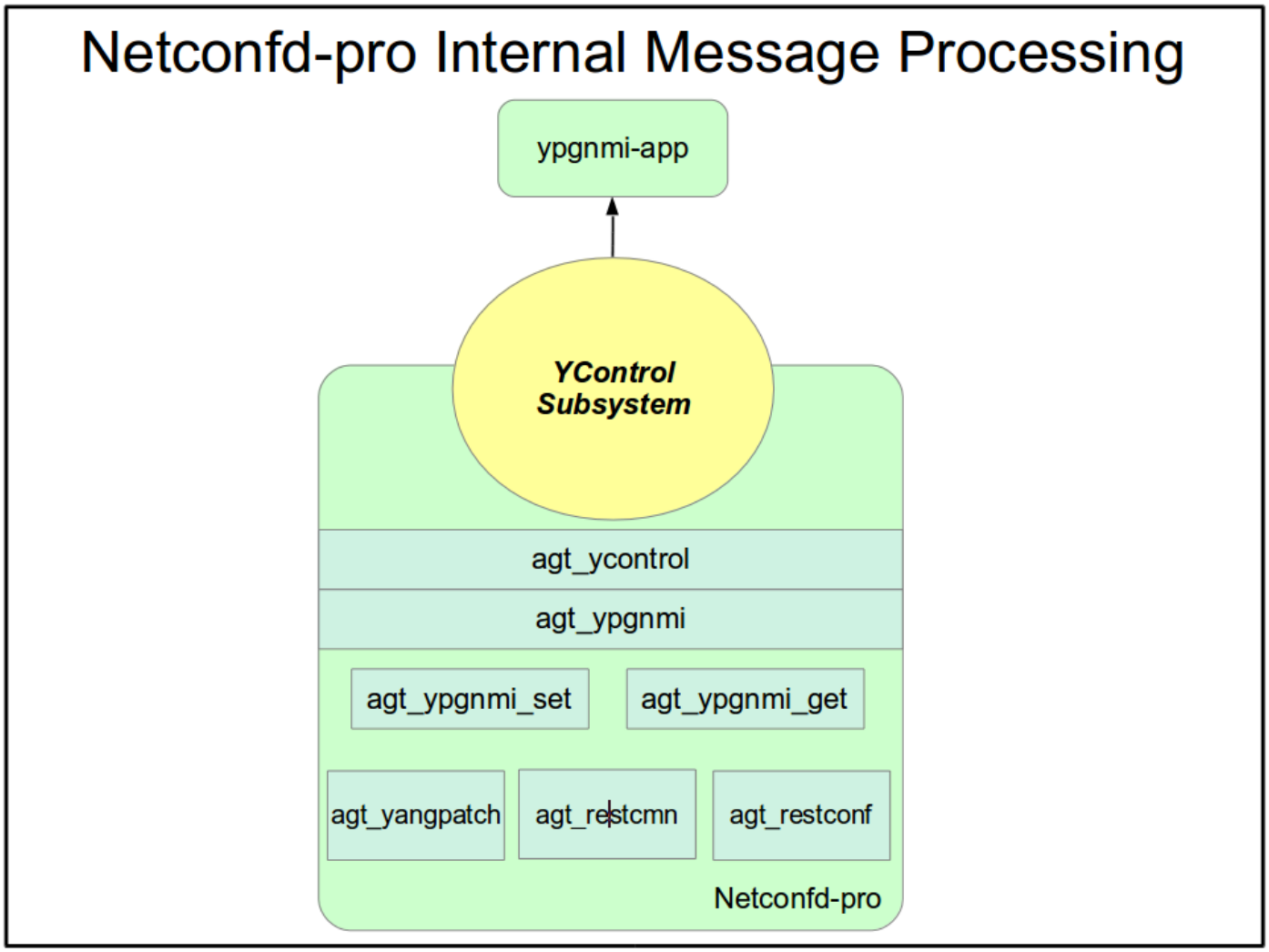
```
        container get-response {
          description
           "Message type: server-response;
            Purpose: Provide the contents of the
            configuration contents.

            Expected Response Message: none";

          container notifications {
            description
             "A set of Notification messages. The target MUST generate a
              Notification message for each path specified in the client's
              GetRequest, and hence MUST NOT collapse data from multiple
              paths into a single Notification within the response.

              The timestamp field of the Notification message MUST be set
              to the time at which the target's snapshot of the relevant
              path was taken.";

            uses notifications;
          }

          uses extension-field;
        }  // container get-response


        container set-request {
          description
           "Message type: subsys-request;
            Purpose: Ask the main server to send the running
            configuration contents

            In response to a SetRequest, the target MUST respond with a
            SetResponse message. For each operation specified in the
            SetRequest message, an UpdateResult message MUST be included
            in the response field of the SetResponse. The order in which
            the operations are applied MUST be maintained such that
            UpdateResult messages can be correlated to the SetRequest
            operations. In the case of a failure of an operation,
            the status of the UpdateResult message MUST be populated with
            error information. In addition, the status of the SetResponse
            message MUST be populated with an error message indicating
            the success or failure of the set of operations within the
            SetRequest message.";


          leaf prefix {
            type string;
            description
              "The prefix specified is applied to all paths defined
               within other fields of the message.";
          }

          leaf-list delete {
            type string;
            description
              "A set of paths which are to be removed from the data tree.
               Where a path is contained within the delete field of the
               SetRequest message, it should be removed from the target's
               data tree. In the case that the path specified is to an
               element that has children, these children MUST be recursively
               deleted. If a wildcard path is utilised, the wildcards
               MUST be expanded by the target, and the corresponding
```

```
            elements of the data tree deleted. Such wildcards MUST
            support paths specifying a subset of attributes required
            to identify entries within a collection (list, array, or map)
            of the data schema.

            In the case that a path specifies an element within the data
            tree that does not exist, these deletes MUST be silently
            accepted.";
        }

        container replace {
          description
            "A set of Update messages indicating elements of the
             data tree whose content is to be replaced.";

          uses update-list;
        }

        container update {
          description
            "A set of Update messages indicating elements of the data
             tree whose content is to be updated.";

          uses update-list;
        }

        uses extension-field;
      }

      container set-response {
        description
          "A set of Update messages indicating elements of the
           data tree whose content is to be replaced.

           Expected Response Message: none";

        leaf prefix {
          type string;
          description
            "The prefix specified is applied to all paths defined
             within other fields of the message.";
        }

        leaf timestamp {
          type string;
          description
            "A timestamp at which the set request message was accepted by
             the system.";
        }

        list update-result {
          key "path op";

          description
            "Containing a list of responses, one per operation specified
             within the SetRequest message. Each response consists of
             an UpdateResult message with the following fields";

          leaf path {
            type string;
            description
              "The path specified within the SetRequest. In the case
               that a common prefix was not used within the SetRequest,
```

```
                the target MAY specify a prefix to reduce repetition of
                path elements within multiple UpdateResult messages in
                the request field.";
            }

            leaf op {
              type enumeration {
                enum invalid;
                enum delete;
                enum replace;
                enum update;
              }

              description
                "The operation corresponding to the path. This value
                 MUST be one of DELETE, REPLACE, or UPDATE.";
            }

            container status {
              leaf code {
                type int32;
                description
                  "code - a 32-bit integer value corresponding to the
                   canonical gRPC error code. The Netconfd Internal
                   error in this message.";
              }
              leaf message {
                type string;
                description
                  "message - a human-readable string describing the
                   error condition. This string is not expected to
                   be machine-parsable, but rather provide contextual
                   information which may be passed to upstream systems.
                   In our case error-string.";
              }
              leaf details {
                type string;
                description
                  "details - a repeated field of protobuf. Any messages
                   that carry error details. In our case just error-info";
              }
            }
          }

          uses extension-field;
        }

      } // choice message-type
    } // container yp-gnmi
  } // augment
}
```

13.3 Netconfd-pro Processing

The following diagram illustrates how the **ypgnmi-app** application requests are getting processed in the **netconfd-pro** server.



The message processing steps:

- Parse an incoming <ycontrol> message from a subsystem
- Handle an incoming <ycontrol> message from a subsystem and dispatch this message to the service handler
- Run callback template for YControl yp-gnmi services: message handler
- Initializer the RESTCONF specific dispatcher for the yp-gnmi request
  - Dispatch RESTCONF GET request in case of gNMI GetRequest
  - Dispatch YANG-PATCH processing in case of gNMI SetRequest
- Construct the Response based on the RESTCONF dispatcher return values
- Send the reply to the yp-gnmi service

13.4 gNMI Service definition

The main gNMI service functionality contains the following requests (RPC):

- **Capabilities** - used by the client and target as an initial handshake to exchange capability information.
- **Get** - used to retrieve snapshots of the data on the target by the client.
- **Set** - used by the client to modify the state of the target.
- **Subscribe** - used to control subscriptions to data on the target by the client.

Future revisions of the gNMI specification MAY result in additional services being introduced, and hence an implementation MUST NOT make assumptions that limit to a single service definition.

The **yp-gnmi** application is capable to handle these requests and transfer them to the **netconfd-pro** server for further processing as well as handle replies from the **netconfd-pro** server and transmit them back to the gNMI clients. The following sections will describe in more details how these requests are handled internally in the **netconfd-pro** and how **yp-gnmi** application transform original gNMI requests and how they are transmitted to and from the **netconfd-pro**.

13.4.1 gNMI GetRequest

The gNMI GET RPC specifies how to retrieve one or more of the configuration attributes, state attributes, derived state attributes, or all attributes associated with a supported mode from a date tree. A GetRequest is sent from a client to the target to retrieve values from the data tree. A GetResponse is sent in response to a GetRequest.

The following example shows a Get Request on JSON structure:

```
XPath: /oc-if:interfaces/interface[name=Loopback111]
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
encoding: JSON_IETF
++++++++ Recevied get response: ++++++++
notification {
  timestamp: 1521699434792345469
  update {
    path {
      elem {
        name: "oc-if:interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
    }
    val {
      json_ietf_val: "{\n\t\"openconfig-interfaces:name\":\t\"Loopback111\",\n\t\
       "openconfig-interfaces:config\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\"\n\t},\n\t\
       "openconfig-interfaces:state\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\
       "openconfig-interfaces:ifindex\":\t52,\n\t\t\
       "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\
       "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\
       "openconfig-interfaces:last-change\":\t2018,\n\t\t\
       "openconfig-interfaces:counters\":\t{\n\t\t\t\
       "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\
```

```
      "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-unknown-protos\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\
      "openconfig-interfaces:last-clear\":\t2018\n\t\t},\n\t\t\
      "openconfig-platform:hardware-port\":\t\"Loopback111\"\n\t},\n\t\
      "openconfig-interfaces:subinterfaces\":\t{\n\t\t\
      "openconfig-interfaces:index\":\t0,\n\t\t\
      "openconfig-interfaces:config\":\t{\n\t\t\t\
      "openconfig-interfaces:index\":\t0,\n\t\t\t\
      "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\t\
      "openconfig-interfaces:enabled\":\t\"true\"\n\t\t},\n\t\t\
      "openconfig-interfaces:state\":\t{\n\t\t\t\
      "openconfig-interfaces:index\":\t0,\n\t\t\t\
      "openconfig-interfaces:name\":\t\"Loopback111.0\",\n\t\t\t\
      "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\t\
      "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\t\
      "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\t\
      "openconfig-interfaces:last-change\":\t2018,\n\t\t\t\
      "openconfig-interfaces:counters\":\t{\n\t\t\t\t\
      "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:last-clear\":\t2018\n\t\t\t\t}\n\t\t},\n\t\t\
      "openconfig-if-ip:ipv6\":\t{\n\t\t\t\
      "openconfig-if-ip:config\":\t\"false\",\n\t\t\t\
      "openconfig-if-ip:state\":\t\"false\"\n\t\t}\n\t}\n}"
    }
  }
}
```

The following example shows a GetRequest on a leaf on JSON structure:

```
XPath: /oc-if:interfaces/interface[name=Loopback111]/state/oper-status
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
```

```
  elem {
    name: "state"
  }
  elem {
    name: "oper-status"
  }
}
encoding: JSON_IETF
++++++++ Recevied get response: ++++++++
notification {
  timestamp: 1521699326012374332
  update {
    path {
      elem {
        name: "oc-if:interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"UP\""
    }
  }
}
```

13.4.2 gNMI SetRequest

The Set RPC specifies how to set one or more configurable attributes associated with a supported model. A SetRequest is sent from a client to a target to update the values in the data tree.

Within an individual transaction (SetRequest) the order of operations is delete, replace, update.

In a SetRequest, only fully-specified (wildcards, and all keys-specified paths are not supported.) paths, and "json_ietf_val" or "json_val" TypedValue are supported. JSON keys must contain a YANG-prefix, in which the namespace of the following element differs from parent. The “routed-vlan” element derived from augmentation in openconfig-vlan.yang must be entered as “oc-vlan:routed-vlan”, because it is different from the namespace of the parent node (The parent node prefix is oc-if.).

The total set of deletes, replace, and updates contained in any one SetRequest is treated as a single transaction. If any subordinate element of the transaction fails; the entire transaction will be disallowed and rolled back. A SetResponse is sent back for a SetRequest.

In the case that any operation within the SetRequest message fails, then, the target MUST NOT apply any of the specified changes, and MUST consider the transaction as failed. The target SHOULD set the status code of the SetResponse message to Aborted (10), along with an appropriate error message, and MUST set the message field of theUpdateResult corresponding to the failed operation to an Error message indicating failure. In the case that the processed operation is not the only operation within the SetRequest the target MUST set the message field of the UpdateResultmessages for all other operations, setting the code field to Aborted (10).

The following example shows a SetRequest on JSON structure:

```
Creating UPDATE update for /oc-if:interfaces/interface[name=Loopback111]/config/
XPath: /oc-if:interfaces/interface[name=Loopback111]/config/
++++++++ Sending set request: ++++++++
update {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
  }
  val {
    json_ietf_val: "{\"config\":{\"openconfig-interfaces:enabled\":\"false\"}}"
  }
}
++++++++ Recevied set response: ++++++++
response {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
```

```
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
  }
  op: UPDATE
}
timestamp: 1521699342123890045
```

The following example shows a SetRequest on leaf on JSON structure:

```
Creating UPDATE update for /oc-if:interfaces/interface[name=Loopback111]/config/description
XPath: /oc-if:interfaces/interface[name=Loopback111]/config/description
++++++++ Sending set request: ++++++++
update {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "description"
    }
  }
  val {
    json_ietf_val: "\"description\":\"UPDATE DESCRIPTION\""
  }
}
++++++++ Recevied set response: ++++++++
response {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
    elem {
```

```
      name: "description"
    }
  }
  op: UPDATE
}
timestamp: 1521699342123890045
```

13.4.3 gNMI JSON_ietf_val

The JSON type indicates that the value is encoded as a JSON string as specified in RFC 7159. Additional types (such as, JSON_IETF) indicate specific additional characteristics of the encoding of the JSON data (particularly where they relate to serialization of YANG-modeled data).

The following is a sample JSON_ietf_val message:

```
val {
  json_ietf_val:"{
   "oc-if:config": {
     "oc-if:description":
       "UPDATE DESCRIPTION"
   }
  }"
}
```

13.4.4 gNMI Error Messages

When errors occur, gNMI returns descriptive error messages. The following section displays some gNMI error messages.

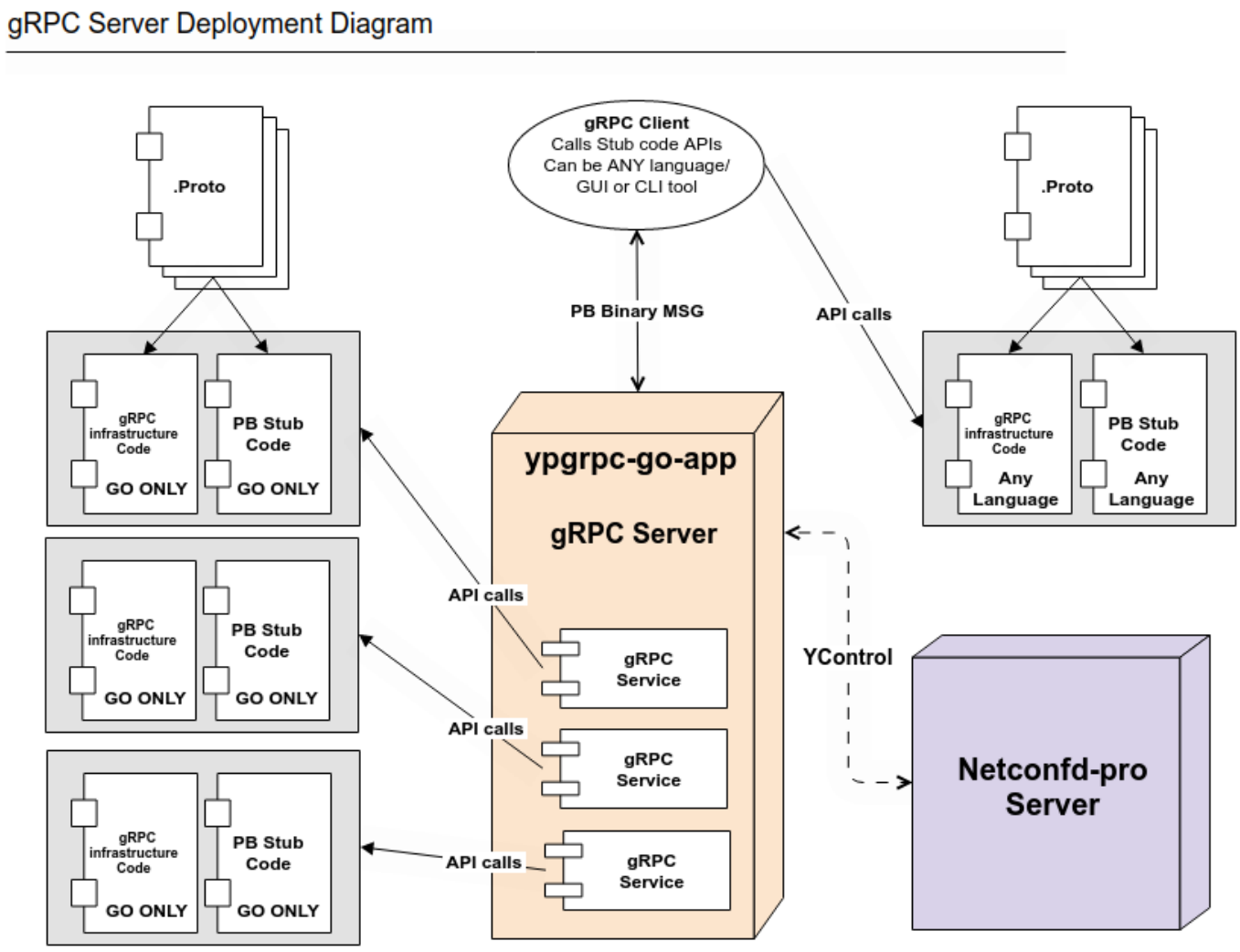
The following sample error message is displayed when the path is invalid:

gNMI Error Response:

```
== getRequest:
path: <
  elem: <
    name: "unknown-resource"
  >
>
encoding: JSON_IETF

Get failed: rpc error: code = Code(385) desc = unknown resource
```

14 YumaPro gRPC Subsystem

This section describes the gRPC integration within the **netconfd-pro** server and **ypgrpc-go-app** application.



The above diagram illustrates deployment of the gRPC server, all its Services and messages handling and how the **netconfd-pro** server is integrated into this deployment.

The gRPC server and **netconfd-pro** server integration is deployed with help of **ypgrpc-go-app** application that transfers information between integrated gRPC server and gRPC clients and **netconfd-pro** server. Also, this is an application that provides faster and easier platform to implement gRPC Services. It is similar to **db-api-app** where users create instrumentation for their Services and RPCs.

14.1 Ypgrpc-go-app Overview

The **ypgrpc-go-app** application is a YControl subsystem that communicates to the **netconfd-pro** server and also it is a gRPC server that communicates to gRPC clients. The main role of the **ypgrpc-go-app** application is to host gRPC server and provide common place to implement and instrument gRPC services and also provide monitoring and control using **netconfd-pro** server.

14.1.1 Ypgrpc-go-app Benefits

The **ypgrpc-go-app** application provides following benefits:

- Common place to implement and instrument .Proto Services and RPCs
- Single gRPC Server to handle
- Subsystem reports to the **netconfd-pro** server with available capabilities
- Subsystem reports when subscriptions start and end to the **netconfd-pro** server for monitoring information
- Remote monitoring and control of gRPC server using **netconfd-pro** server
- Possibility to remotely shutdown the gRPC server using **netconfd-pro** server

14.1.2 Ypgrpc-go-app Processing

The **ypgrpc-go-app** application is written in GO language and talks to the **netconfd-pro** server via socket and acts as a YControl subsystem (similar to **db-api-app**).

gRPC clients can be written in any languages that are supported for gRPC clients. The client part is out of the scope of this document and the current gRPC protocol integration does not include client part. The clients communicate to the gRPC server with help of the **ypgrpc-go-app** application and send gRPC request to the application.

The core of the **ypgrpc-go-app** is the gRPC server and integrated stub code from auto generated files from .proto files:

- Handle integrated stub code callbacks. Callbacks that are integrated from stub code that were generated from .Proto files using **protoc** tool
- Register gRPC server for the protobuf message handling and gRPC Services callback invocation
- Run main Serve loop that handles all the client/server communication

The processing between gRCP client to the **netconfd-pro** server can be split into following components:

- **gRCP clients to the ypgrpc-go-app processing:** includes message parsing, gRPC Services callback invocation
- **The ypgrpc-go-app application to the netconfd-pro processing:** includes YControl messages exchange and stream information exchange when a new stream opens or closes
- **The netconfd-pro server internal processing**: includes subsystem registration, subsystem messages handling and parsing, gRPC monitoring information handling (gRPC server and streams status).

The **ypgrpc-go-app** implements multiple goroutines to manage the replies and the clients. All of this managers are goroutines. They run in parallel and asynchronously. The following goroutines are implemented in the **ypgrpc-go-app:**

- **Reply Manager goroutine.** This manager is responsible for any already parsed messages from the **netconfd-pro** server or gRPC client, it stores any not processed messages that are ready to be processed.

- **Message Manager goroutine.** This manager is responsible for storing any ready to be processed messages that are going to the **netconfd-pro** server and that are coming back from the server.

14.1.3 Startup Procedure

The **ypgrpc-go-app** application has the following startup steps for the gRPC server component:

- Initialize all the prerequisites and parse all the CLI parameters
- Open TCP socket to listen for clients requests
- Serve any incoming gRPC messages from gRPC clients and send **open-stream-event** or **close-stream-event** to the **netconfd-pro** server if needed with help of all the goroutine managers.

The **ypgrpc-go-app** acts as a YControl subsystem (similar to **db-api-app**), however, it does not terminate after one edit or get request. Instead it continuously listens to the **netconfd-pro** server and keeps the AF_LOCAL or TCP socket open to continue communication whenever it's needed. The communication is terminated only if the **ypgrpc-go-app** application is terminated, the **netconfd-pro** server terminates, or the **netconfd-pro** sends the request to terminate the **ypgrpc-go-app** application. All the message definitions described in the **yumaworks-yp-grpc.yang** YANG module.

The **ypgrpc-go-app** application has the following startup steps to initialize connection with the **netconfd-pro** server:

- Initialize all the prerequisites and parse all the CLI parameters
- Based on the **--proto** CLI parameter load all the Proto files and create capability structure for provided proto files
- Open socket and send <ncx-connect> request to the server with
  - transport = netconf-aflocal
  - protocol = yp-grpc
- Register yp-grpc service
  - Send <register-request> to the server
  - Register **ypgrpc-go-app** subsystem and initialize all corresponding code in the **netconfd-pro** server to be ready to handle **ypgrpc-go-app** application requests
- Send **capability-ad-event** message to the **netconfd-pro** server to advertise all available Services, Methods and streams
- Keep listening socket until terminated

14.1.4 Running Ypgrpc-go-app

Run the server with the setting **--with-grpc=true** flag as follows:

```
mydir> sudo netconfd-pro --log-level=debug4 --with-grpc=true \
       -- fileloc-fhs=true
```

Start the **ypgrpc-go-app** application. Note that you have to provide your certificates to start the application:

```
mydir> man ypgrpc-go-app
mydir> ypgrpc-go-app --cert=~/certs/server.crt --ca=~/certs/ca.crt \
       --key=~/certs/server.key \
       --fileloc-fhs \
       --protopath=$HOME/protos \
       --proto=helloworld --proto=example
```

OR run it in “insecure” mode for test or verification:

```
mydir> ypgrpc-go-app --log-level=debug --fileloc-fhs --insecure \
       --protopath=$HOME/protos \
       --proto=helloworld --proto=example
```

After this step the gRPC server starts to listen for any gRPC client requests and will handle all the request for the provided example.proto and helloworld.proto .proto files and will advertise gRPC capabilities and example.proto, helloworld.proto Services to the **netconfd-pro** server.

14.1.5 Closing Ypgrpc-go-app

The **ypgrpc-go-app** can be shut down by typing Ctrl-C in the window that started the application.

If the **netconfd-pro** server is not running when **ypgrpc-go-app** is started the application will terminate with an error message stating that the **netconfd-pro** server is not running.

If the **netconfd-pro** server is shut down then **ypgrpc-go-app** will also shutdown.

The **netconfd-pro** server has <grpc-shutdown> NETCONF operation that can be triggered to shut down the **ypgrpc-go-app** application.

14.2 Ypgrpc-go-app Message Format

The **ypgrpc-go-app** application uses several messages to interact with the **netconfd-pro** server.

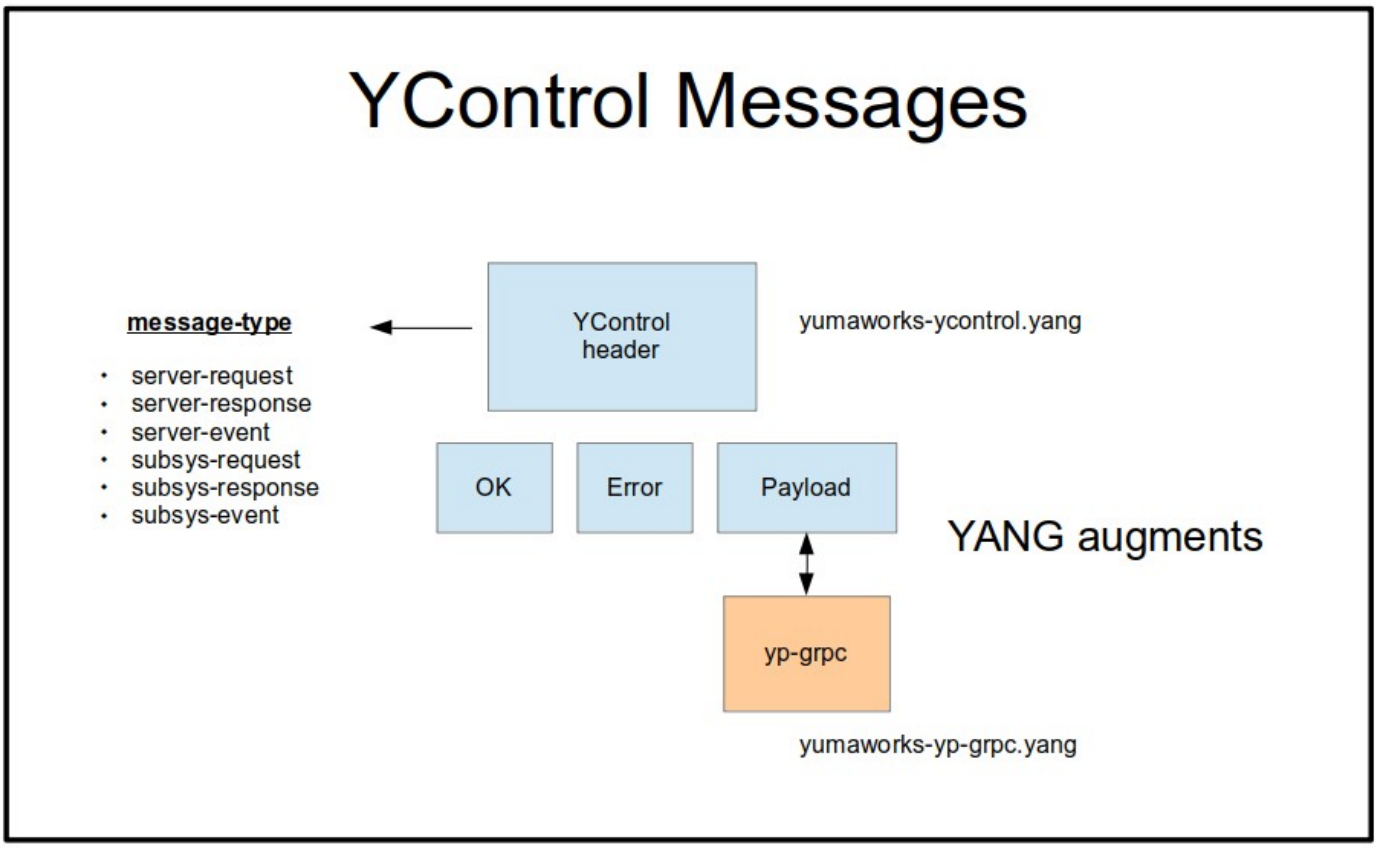
14.2.1 YControl Integration

The **ypgrpc-go-app** application and YControl subsystem service that is part of the **netconfd-pro** have the following messages interaction:

- **capability-ad-event**: **ypgrpc-go-app** sends subsystem event to advertise all the gRPC available and active capabilities during the registration time
- **open-stream-event**: **ypgrpc-go-app** sends subsystem event to advertise a new gRPC server or client stream(s)
- **close-stream-event**: **ypgrpc-go-app** sends subsystem event to advertise that the gRPC server or client stream was closed.

These messages are defined in the yumaworks-yp-grpc YANG module. The **ypgrpc-go-app** payload is defined as a YANG container that augments the YControl “message-payload” container.

The following diagram illustrates the YControl messages overview:



14.2.2 Ypgrpc-go-app Registration Message Flow

During the startup phase the server will initialize the **yp-grpc** subsystem callback functions and handlers (similar way as for **db-api** module does).

The connection with the server is getting started with <ncx-connect> message that adds the YControl subsystem with the "example-grpc" subsystem ID to the server (**agt_connect** module).

YControl protocol connection parameters:

- transport: **netconf-aflocal**
- protocol: **yp-grpc**
- <port-num> not used

Additional parameters:

- subsys-id: **example-grpc**

The Registration message flow looks as follows:

- **Ypgrpc-go-app to Server:** *register-request:*
  The **ypgnmi-app** service registers callbacks supported by the subsystem.
- **Server to ypgrpc-go-app:** *ok:*
  The server responds to the register request with an <ok> or an <error> message
- **Ypgrpc-go-app to Server:** *capability-ad-event:*
  Sends subsystem event to advertise all the gRPC available and active capabilities during the registration time

The module "yumaworks-yp-grpc.yang" defines all the messages

14.2.3 Yumaworks-yp-grpc YANG Module

The YP-gRPC subsystem service messages are defined in the **yumaworks-yp-grpc.yang** module.

```
module yumaworks-yp-grpc {
  yang-version 1.1;
  namespace "http://yumaworks.com/ns/yumaworks-yp-grpc";
  prefix "ypgrpc";

  import yumaworks-ycontrol { prefix yctl; }
  import ietf-yang-types { prefix yang; }
  import ietf-inet-types { prefix inet; }

  organization "YumaWorks, Inc.";

  contact
    "Support <support@yumaworks.com>";

  description
    "YumaPro gRPC Application message definitions.

     Copyright (c) 2014 - 2021 YumaWorks, Inc. All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the BSD 3-Clause License
     http://opensource.org/licenses/BSD-3-Clause";

  revision 2021-08-23 {
    description
      "Initial version";
  }


  augment "/yctl:ycontrol/yctl:message-payload/yctl:payload/yctl:payload" {
    container yp-grpc {

      choice message-type {
        mandatory true;

        case register-request-case {
          leaf register-request {
            type empty;
            description
              "Message type: subsys-request;
               Purpose: register the YP-gRPC subsystem
               Expected Response Message: ok or error";
          }
        }

        container capability-ad-event {
          description
            "Subsystem event to advertise all the gRPC
             available and active capabilities during
             the registration time.
             Type: subsys-event
             Expected Response Message: none";

          leaf name {
```

```
            mandatory true;
            type yang:yang-identifier;
            description
              "Name of the gRCP server.";
          }

          leaf address {
            type inet:host;
            mandatory true;
            description
              "IP Address or host name for the gRPC server.
               The value returned is implementation specific
               (E.g., hostname, IPv4 address, IPv6 address)";
          }

          leaf port {
            type inet:port-number;
            description
              "TCP port number for the gRPC server.
               If not present then the default port for
               the protocol will be used.";
          }

          leaf-list proto {
            type string;
            description
              "The list of proto files that gRPC server supports.";
          }

          list service {
            key name;
            description
              "List of gRPC Services supported by the gRPC server and
               related information.";

            leaf name {
              type string;
              description
                "Name of the gRCP Service associated with this list entry.";
            }

            list method {
              key name;
              description
                "The list of methods supported by the gRPC server and
                 related information.";

              leaf name {
                type string;
                description
                  "Name of the Service Method associated with this list entry.";
              }
              leaf client-streaming {
                type boolean;
                description
                  "True if the client supports streaming for this method
                   FALSE otherwise.";
              }
              leaf server-streaming {
                type boolean;
                description
                  "True if the server supports streaming for this method.
                   FALSE otherwise.";
```

```
        }
      }
    }
  }

  container open-stream-event {
    description
      "Subsystem event to advertise a new gRPC
       server or client stream(s).
       Type: subsys-event
       Expected Response Message: none";

    list server-stream {
      key name;
      description
        "Contains a list of open server gRPC streams.";

      leaf name {
        type string;
        description
          "Name of a gRCP server stream.";
      }

      leaf location {
        type inet:uri;
        mandatory true;
        description
          "Contains a URL that represents the RPC that uses
           this client stream.";
      }
    }

    list client-stream {
      key name;
      description
        "Contains a list of open client gRPC streams.";

      leaf name {
        type string;
        description
          "Name of a gRCP client stream.";
      }

      leaf location {
        type inet:uri;
        mandatory true;
        description
          "Contains a URL that represents the RPC that uses
           this server stream.";
      }
    }
  }

  container close-stream-event {
    description
      "Subsystem event to advertise that the gRPC server or
       client stream was closed.
       Type: subsys-event
       Expected Response Message: none";

    leaf-list server-stream {
      type string;
      description
```

```
              "The list of closed server gRPC streams.";
          }

          leaf-list client-stream {
            type string;
            description
              "The list of closed client gRPC streams.";
          }
        }
      }
    }
  }
}
```

15 Automation Control

The YANG language includes many ways to specify conditions for database validity, which traditionally are only documented in DESCRIPTION clauses. The YANG language allows vendors and even data modelers to add new statements to the standard syntax, in a way that allows all tools to skip extension statements that they do not understand.

The **yangdump-pro** YANG compiler sames all the non-standard language statements it finds, even those it does not recognize. These are stores in the **ncx_appinfo_t** data structure in **ncx/ncxtypes.h**..

There are also SIL access functions defined in **ncx/ncx_appinfo.h** that allow these language statements to be accessed. If an argument string was provided, it is saved along with the command name.

Several data structures contains an 'appinfoQ' field to contain all the ncx_appinfo_t stuctures that were generated within the same YANG syntax block (E.g., within a typedef, type, leaf, import statement).

15.1 YANG Parser Object Template APIs

The YANG compiler built into the netconfd-pro server has some yp-system level callback APIs that allow YANG data structures to be customized to assist SIL automation or improve processing performance.

15.1.1 Object Template User Flags

There is a special field in the **obj_template_t** structure to allow vendor-specific flag definitions for an object template. This field is 32 bits wide and initialized to zero when the object template is created.

```
Field definition:  uint32 uflags;

Access Macro: OBJ_USER_FLAGS(obj)
```

15.1.2 Object Template Callback API

This feature allows the object template for YANG data nodes to be examined, and the user flags to be set as needed.

This API is called for all objects during YANG module parsing. The callback may wish to use filters such as **obj_is_data_db(obj)** to limit scanning to only database objects.

The API template is defined in **ncx/ncxtypes.h**:

```
/* user function callback template when a YANG object is
 * parsed by yang_obj.c. This API is invoked at the
 * end of the resolve phase if the status is NO_ERR
 * It is skipped if the object has errors detected at the time
 *
 * ncx_yang_obj_cbfn_t
 *
 *  Run an instrumentation-defined function
 *  for a 'object parsed' event
 *
 * INPUTS:
 *   mod == module that is being parsed now
 *   obj == object being parsed
 */
typedef void
    (*ncx_yang_obj_cbfn_t) (ncx_module_t *mod,
                            struct obj_template_t_ *obj);
```

The registration function needs to be called from the yp-system initialization callbacks. This callback should be installed before any SIL-related YANG modules are loaded.

The registration functions are defined in **ncx/ncx.h**:

```
/********************************************************************
* FUNCTION ncx_set_yang_obj_callback
*
* Set the callback function for a YANG object parse event
*
* INPUT:
*   cbfn == callback function to set
*
* RETURNS:
*  status
*********************************************************************/
extern status_t
    ncx_set_yang_obj_callback (ncx_yang_obj_cbfn_t cbfn);


/********************************************************************
* FUNCTION ncx_clear_yang_obj_callback
*
* Clear the callback function for a parse-object event
*
* INPUT:
*   cbfn == callback function to find and clear
*
```

```
*********************************************************************/
extern void
    ncx_clear_yang_obj_callback (ncx_yang_obj_cbfn_t cbfn);
```

15.1.3 YANG Object template Callback Usage Example

This example callback function checks for a proprietary YANG extension within a YANG object.

```
/***********  Example YANG Object Template Callback **********/

/*
 * Assume YANG module foo exists with extension acme1
 *
 * module foo {
 *   prefix f;
 *   ...
 *   extension acme1 { ... }
 *
 * The extension is used inside object definitions. E.g:
 *
 *    leaf X {
 *      f:acme1;
 *      type string;
 *    }
 *
 * Assume there is a vendor bit defined for the user flags field
 */

#define FL_ACME_1  0x1
```

Callback function:

```
/* user function callback template when a YANG object is
 * parsed by yang_obj.c. This API is invoked at the
 * end of the resolve phase if the status is NO_ERR
 * It is skipped if the object has errors detected at the time
 *
 * ncx_yang_obj_cbfn_t
 *
 *  Run an instrumentation-defined function
 *  for a 'object parsed' event
 *
 * INPUTS:
 *   mod == module that is being parsed now
 *   obj == object being parsed
 */
void
    example_obj_template_cbfn (ncx_module_t *mod,
                               struct obj_template_t_ *obj)
{
    /* optional: use the module to check certain module names to
     * pre-filter the callback
     */
```

```
    (void)mod;

    /* get the appinfoQ for the object */
    dlq_hdr_t *appinfoQ = obj_get_appinfoQ(obj);
    if (appinfoQ == NULL) {
        return;   // error!
    }

    /* check the object template appinfoQ to see if the vendor
     * extensions are present
     */
    ncx_appinfo_t *appinfo =
        ncx_find_appinfo(appinfoQ,
                         (const xmlChar *)"f",
                         (const xmlChar *)"acme1");
    if (appinfo) {
        OBJ_USER_FLAGS(obj) |= FL_ACME_1;
    }

}
```

Registration inside **yp_system_init1**:

```
#include “ncx.h”

status_t yp_system_init1 (boolean pre_cli)
{
    status_t res = NO_ERR;
    log_debug("\nyp_system init1\n");

    if (pre_cli) {
       ;
    } else {

        /* example -- Register a YANG Object Template Callback */
        res = ncx_set_yang_obj_callback(example_obj_template_cbfn);


    }
    return res;

}  /* yp_system_init1 */
```

15.2 YANG Parser Extension Statement APIs

The YANG “extension” statement allows external language statements to be added to YANG. These external statements follow the same generic structure as all YANG statements. A YANG compiler must be able to skip over external statements. Only tools which claim conformance to an extension are required to enforce the semantics of the extension.

The YANG compiler supports user callbacks to handle specific extension statements. There are two types of callbacks:

1. **Top-level extensions:** These extensions have the module statement itself as its parent. These extensions may be wrappers for additional statements, such as the reserved extension “yang-data”. The callback for these extensions is invoked during the module parsing phase (E.g,., **consume_extension**). **The callback is responsible for completing the parsing phase and consume all tokens for the external statement, including the final semi-colon or right brace.**
2. **Nested extensions:** These extensions are contained within nested statements, which are not at the top-level. They are expected to be simple statements and not wrappers for YANG statements. The callback for these extensions is invoked during the module validation phase (E.g., **resolve_extension**). This callback is not responsible for parsing any module token input. All parsing has already been completed

15.2.1 YANG Extension Handler Callback

The following callback API is defined in **ncx/ext.h**:

```
/* One YANG Extension Handler Callback
 *
 * ext_cbfn_t
 *
 * Handle the parsing and processing of an external statement
 * using the associated YANG extension statement
 *
 * This callback is invoked when the external statement is
 * first encountered. The current token is the argument string
 * if any or the identifier token if none.
 * The next token is expected to be a semi-colon or a left brace
 * The callback is expected to parse the closing semi-colon or
 * entire sub-section including starting brace
 *
 * INPUTS:
 *   rawpcb == parser control block in progress (cast as void *)
 *   mod == module being processed
 *   tkc == token chain of module tokens parse in progress
 *   ext == extension definition record (allows a handler to
 *          process multiple extension types)
 *   cookie == cbfn_cookie from the extension 'ext'
 *   arg == argument string used in the external statement (if any)
 *   node_type == type of node being processed; direct parent
 *          statement of the external statement using the extension
 *          If NULL, then the parent statement is the module itself,
 *          and 'mod' should be used as the 'node' pointer
 *   node == pointer to node indicated by node_type
 * OUTPUTS:
 *
 * RETURNS:
 *   status of processing
 */
```

```
typedef status_t (*ext_cbfn_t)
    (void *rawpcb,    // struct yang_pcb_t_ *pcb
     ncx_module_t *mod,
     tk_chain_t *tkc,
     struct ext_template_t_ *ext,
     void *cookie,
     const xmlChar *arg,
     ncx_node_t node_type,
     void *node);
```

The registration function is defined in **ncx/etc.h**. This API should be registered in the yp-system initialization phase, before YANG modules are parsed.

```
/********************************************************************
* FUNCTION ext_register_cbfn
*
* Register a callback function for the specified extension
* If multiple callbacks for same extension, then last one wins
*
* INPUTS:
*   modname == module name defining the extension
*   extname == extension name
*   cbfn == pointer to callback function to register
*   cbfn_cookie == optional cookie to register; will not be
*            freed when the extension is freed
*            Use macro EXT_CBFN_COOKIE(ext) to access from callback
*              == NULL if not used
*********************************************************************/
extern status_t
    ext_register_cbfn (const xmlChar *modname,
                       const xmlChar *extname,
                       ext_cbfn_t cbfn,
                       void *cbfn_cookie);
```

15.2.2 Usage Example

This example callback function checks for a proprietary YANG extension within a YANG object.

It uses the same YANG module as the YANG Object Template Callback.

```
/*
 * Callback is invoked to check a specific extension in an
 * obj_template_t, typ_template_t, typ_def_t
 *
 * Assume the same YANG module foo exists with extension acme1
 *
 * The example callback does the same task as the
 * example_obj_template_cbfn, using the per-callback approach
 */

/* One YANG Extension Handler Callback
 *
 * example_ext_cbfn
 *
 * Handle the parsing and processing of an external statement
 * using the associated YANG extension statement
 *
 * This callback is invoked when the external statement is
 * first encountered. The current token is the argument string
 * if any or the identifier token if none.
 * The next token is expected to be a semi-colon or a left brace
 * The callback is expected to parse the closing semi-colon or
 * entire sub-section including starting brace
 *
 * INPUTS:
 *   rawpcb == parser control block in progress (cast as void *)
 *   mod == module being processed
 *   tkc == token chain of module tokens parse in progress
 *   ext == extension definition record (allows a handler to
 *          process multiple extension types)
 *   cookie == cbfn_cookie from the extension 'ext'
 *   arg == argument string used in the external statement (if any)
 *   node_type == type of node being processed; direct parent
 *          statement of the external statement using the extension
 *          If NULL, then the parent statement is the module itself,
 *          and 'mod' should be used as the 'node' pointer
 *   node == pointer to node indicated by node_type
 * OUTPUTS:
 *
 * RETURNS:
 *   status of processing
 */
static status_t
     example_ext_cbfn (void *rawpcb,  // struct yang_pcb_t_ *pcb
                       ncx_module_t *mod,
                       tk_chain_t *tkc,
                       struct ext_template_t_ *ext,
                       void *cookie,
                       const xmlChar *arg,
                       ncx_node_t node_type,
                       void *node)
{
    (void)rawpcb;
```

```
    (void)mod;
    (void)tkc;
    (void)ext;
    (void)cookie;
    (void)arg;

    /* ignore this extension in all contexts except object template */
    if (node_type != NCX_NT_OBJ) {
        return NO_ERR;
    }

    /* get the object template */
    obj_template_t *obj = (obj_template_t *)node;

    /* set the acme1 bit */
    OBJ_USER_FLAGS(obj) |= FL_ACME_1;

    return NO_ERR;
}
```

Registration inside **yp_system_init1**:

```
#include “ext.h”

status_t yp_system_init1 (boolean pre_cli)
{
    status_t res = NO_ERR;
    log_debug("\nyp_system init1\n");

    if (pre_cli) {
       ;
    } else {

        /* example -- Register a Extension Handler Callback */
        res = ext_register_cbfn((const xmlChar *)"acme-ext",
                                (const xmlChar *)"acme1",
                                example_ext_cbfn,
                                NULL);   // cookie

    }
    return res;

}  /* yp_system_init1 */
```

15.3 Abstract YANG Data APIs

YANG can be used to define abstract data that can be used within server SIL callbacks and system callbacks. This sort of data is commonly used to read/write files containing YANG data or protocol messages defined in YANG.

This functionality is already supported with the **ncx:abstract** extension, but this method of defining abstract data is more standards-compliant, so it should be used instead of **ncx:abstract.**

15.3.1 rc:yang-data

The RESTCONF Protocol (RFC 8040) includes the YANG extension definition for yang-data. It is used within ietf-restconf.yang and ietf-yang-patch.yang. Any module can define YANG data structures;

Restrictions: YANG does not allow multiple top-level nodes with the same name to be defined in the same module. Make sure that the top-level nodes defined within the yang-data statement do not conflict with other top-level object names. The data defined within the yang-data statement must be 1 container statement or 1 uses statement .

```
module foo {
   // header
   import ietf-restconf { prefix rc; }

   rc:yang-data test1 {
      container test1 {
        leaf leaf1 { type string; }
        leaf leaf2 { type int32; }
      }
   }
}
```

15.3.2 yd:augment-yang-data

The yang-data-ext.yang module defines a YANG extension to allow yang-data nodes to be augmented from a different module. This is not supported by the RESTCONF standard, just supported in netconfd-pro. The plain augment statement can be used instead of this extension, but this should be avoided because a standard YANG compiler is not required to support the RESTCONF yang-data extension (so it will not find the augmented node)

```
module bar {
   // header
   import yang-data-ext { prefix yd; }
   import foo { prefix f; }

   yd:augment-yang-data /f:test1 {
      leaf test2 { type string; }
   }
}
```

15.4 SIL Language Extension Access Functions

The following table highlights the SIL functions in **ncx/ncx_appinfo.h** that allow SIL code to examine any of the non-standard language statements that were found in the YANG module:

Language Extension Access Functions

Function	Description
ncx_find_appinfo	Find an **ncx_appinfo_t** structure by its prefix and name, in a queue of these entries.
ncx_find_next_appinfo	Find the next occurrence of the specified **ncx_appinfo_t** data structure.
ncx_clone_appinfo	Clone the specified **ncx_appinfo_t** data structure.

15.5 Built-in YANG Language Extensions

There are several YANG extensions that are supported by YumaPro. They are all defined in the following YANG files:

- netconfcentral/yuma-ncx.yang
- netconfcentral/yuma-nacm.yang
- ietf/ietf-netconf-acm.yang
- yumaworks/yumaworks-extensions.yang

These YANG extensions are used to 'tag' YANG definitions for some sort of automatic processing by YumaPro programs. Extensions are position-sensitive, and if not used in the proper context, they will be ignored. A YANG extension statement must be defined (somewhere) for every extension used in a YANG file, or an error will be occur.

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

The following sections describe the supported YANG language extensions. All other YANG extension statements will be ignored by YumaPro, if encountered in a YANG file.

15.5.1 ncx:abstract

The **ncx:abstract** extension is used with object definitions to indicate that they do not represent CLI or NETCONF configuration database data instances. Instead, the node is simply an object identifier, an 'error-info' extension, or some other abstract data structure.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
container server-hello {
   description "Generic Server Hello Message Parameters.";

   uses NcCapabilities;

   leaf session-id {
      type session-id-type;
      config false;
   }

   ncx:hidden;
   ncx:abstract;
 }
```

15.5.2 ywx:alt-name

The **ywx:alt-name** extension is used within a data node definition to specify an alternate name for the node. The **--alt-names** parameter must be enabled for these names to be used.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument:

- name: alternate YANG identifier name string to use for the object

Example:

```
leaf system-reset-time {
  ywx:alt-name last-reset;
  type yang:date-and-time;
  config false;
}
```

15.5.3 ncx:cli

The **ncx:cli** extension is used within a container definition to indicate it is only used as a conceptual container for a set of CLI parameters. A top-level container containing this extension will not be included in any NETCONF configuration databases.

Only the following types of YANG objects are allowed within the CLI container at this time:

- leaf
- leaf-list
- choice
- case
- uses (if resolves to only the allowed node types)

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
container yangcli-pro {
   ncx:cli;
   // leafs and choices of leafs inserted here
}
```

15.5.4 ywx:cli-text-block

The **ywx:cli-text-block** extension is used to force CLI text syntax within a container. It is only used by yangcli-pro for test-suite 'setup' and 'cleanup' sections at this time.

If this extension is present in an empty container or list, it will be treated in unit-test parsing as a container or list of ordered text commands, 1 per line. Line extension is needed to wrap a command into many lines.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: none

Example:

```
 container setup {
    ywx:cli-text-block;
 }

Example test script or conf file usage:

 setup {
   run test1-script
   get-config source=running
   lock target=candidate
   some-long-command parms='this is a wrapped \
     line in a text block'
 }
```

15.5.5 ywx:datapath

The **ywx:datapath** extension is used by the YControl subsystem to specify the real object to use for parsing an “anyxml” or “container” node as a different object type.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: path string

Example:

```
"Used within a container or anyxml definition to indicate
that the object path for the data node should be sent
in the value as an attribute.  The SIL-SA parser will use
the datapath attribute to select the object template
to use for parsing, instead of generic anyxml.

   anyxml newval {
     ywx:datapath;
   }
   anyxml curval {
     ywx:datapath;
   }

   If /foo/bar/leaf2 is edited, the <edit> message
   will be generated with the datapath attribute
   from the yumaworks-attrs module.

    <newval ywattrs:datapath='/foo/bar/leaf2'>42</newval>
    <curval ywattrs:datapath='/foo/bar/leaf2'>67</curval>
```

15.5.6 nacm:default-deny-all

The **nacm:default-deny-all** extension is used by the to indicate that the data model node represents a sensitive security system parameter.

If present, and the NACM module is enabled (E.g., **/nacm/enable-nacm** object equals 'true'), the NETCONF server will only allow the designated 'recovery session' to have read, write, or execute access to the node. An explicit access control rule is required for all other users. The 'default-deny-write' extension MAY appear within a data definition statement. It is ignored otherwise.

YANG Files: (only 1 can be active on the server, selected in **agt_profile**)

- **netconf/modules/ietf/ietf-netconf-acm.yang**
- **netconf/modules/netconfcentral/yuma-nacm.yang**

Argument: none

Example:

```
rpc shutdown {
  nacm:default-deny-all;
}
```

15.5.7 nacm:default-deny-write

The **nacm:default-deny-write** extension is used by the to indicate that the data model node represents a sensitive security system parameter.

If present, and the NACM module is enabled (E.g., **/nacm/enable-nacm** object equals 'true'), the NETCONF server will only allow the designated 'recovery session' to have write access to the node. An explicit access control rule is required for all other users. The 'default-deny-write' extension MAY appear within a data definition statement. It is ignored otherwise.

YANG Files: (only 1 can be active on the server, selected in **agt_profile**)

- **netconf/modules/ietf/ietf-netconf-acm.yang**
- **netconf/modules/netconfcentral/yuma-nacm.yang**

Argument: none

Example:

```
leaf enable-system {
  nacm:default-deny-write;
  type boolean;
}
```

15.5.8 ncx:default-parm

The **ncx:default-parm** extension is used by the yangcli-pro program to select a default parameter for the specified RPC input section. It can be used within a CLI container or rpc definition to specify leaf parameter within the CLI container or rpc input section, that is used as the default if no parameter name is entered.

These values must not begin with a dash (-) or double dash (--) sequence or they will be mistaken for CLI parameter names.

This option is somewhat risky because any unrecognized parameter without any prefix (- or --) will be tried as the default parameter type, instead of catching the unknown parameter error. It can also be useful though, for assigning file name parameters through shell expansion, or if there is only one parameter.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument:

- **parm:** name of the leaf to use as the default parameter

Example:

```
rpc connect {
  description "Connect to a NETCONF server.";
  input {
    ncx:default-parm server;

    uses ConnectParms {
      refine user {
       mandatory true;
      }
      refine server {
       mandatory true;
      }
      refine password {
       mandatory true;
      }
    }
 }

 // note that the server parameter can be omitted
 yangcli-pro>  connect localhost user=andy password=yang-rocks
```

15.5.9 ncx:default-parm-equals-ok

The **ncx:default-parm-equals-ok** extension is used by the yangcli-pro program to select a default parameter for the specified RPC input section. It is used within a CLI container or rpc definition to specify a leaf parameter within the CLI container or rpc input section, that is used as the default if no parameter name is entered.

This can be used in addition to **ncx:default-parm** to allow an equals sign '=' in the default **parm** string value.

This option is quite risky because any unrecognized parameter without any prefix (- or --) will be tried as the default parameter type, instead of catching the unknown parameter error. This includes strings containing an equals sign, so an unknown parameter error will never be generated.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
        rpc foo {
          input {
            ncx:default-parm a;
            ncx:default-parm-equals-ok;
            leaf a { type string; }
            leaf b { type int32; }
          }
        }

        yangcli> foo bogus-parm=fred

    This will interpreted as if parameter 'a' were entered:

        yangcli> foo a='bogus-parm=fred'
```

15.5.10 ywx:get-filter-element-attributes

The **ietf:get-filter-element-attributes** extension is defined in ietf-netconf.yang.

This is a reserved extension name, used to define the filter and type attributes used in the <get> operation. It cannot be used in other YANG modules.

15.5.11 ywx:exclusive-rpc

The **ywx:exclusive-rpc** extension is defined in yumaworks-extensions.yang.

It can be used within an rpc definition statement to indicate that the RPC is not allowed to be called concurrently by different sessions. The server will return an **in-use** error if another session is currently invoking the RPC operation and this extension is present in the rpc-stmt.

```
rpc reset-system {
   ywx:exclusive-rpc;
   // only allow 1 session at a time to reset system
}
```

15.5.12 ywx:help

The **ywx:help** extension is used to define a help text string for CLI.

It can be defined within a rpc or data definition statement to provide a short help text string for CLI and other applications to use in addition to the description statement.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument:

- helptext: The help text string which should be 60 characters or less in length.

Example:

```
leaf mtu {
   ywx:help “Maximum transmission unit size”;
   type uint32;
   description “... long description …”;
}
```

15.5.13 ncx:hidden

The **ncx:hidden** extension is used to prevent publication of a YANG data object. It will be ignored for typedefs and other constructs. If present within a data object, that node and any sub-nodes will be ignored when generating HTML documentation or YANG output.

The **yangdump-pro** -f=copy mode is not be affected by this extension.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
container struct {
    ncx:hidden;
    ncx:abstract;
 }
```

15.5.14 ncx:metadata

The **ncx:metadata** extension is used to define an XML attribute to be associated with a data-def-stmt node. Only optional metadata can be defined. Errors for missing XML attributes (except as specified by the YANG language) will not be checked automatically.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument:

- **syntax-string:** syntax of the meta-data attribute
  The syntax string has the following format:

  [prefix:]typename attribute-name

  Any YANG typedef of builtin type can be specified as the type name, except 'empty'.

Example:

```
  container rpc-reply {
    description "Remote Procedure Call response message";

    reference "RFC 4741, section 4.2";

    uses RpcReplyType;

    // do not treat missing message-id as an error
    ncx:metadata  "MessageId message-id";
    ncx:abstract;
  }
```

15.5.15 ncx:no-duplicates

The **ncx:no-duplicates** extension is used to indicate that no duplicate values are allowed in an **ncx:xsdlist** leaf or leaf-list object.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
leaf number-list {
   type string {
     ncx:xsdlist int32;
     ncx:no-duplicates;
   }
}

// the YANG string allowed is a whitespace-separated of numbers
// and none of the numbers can be repeated

<number-list>32 1 -6 103</number-list>
```

15.5.16 ncx:password

The **ncx:password** extension is used to indicate the data type for the leaf is really a password. Only the encrypted version of the password is allowed to be generated in any output.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
leaf system-password {
  ncx:password;
  type string {
     length “8 .. 16”;
  }
}
```

15.5.17 ncx:qname

The **ncx:qname** extension is used to indicate that the content of a data type is a Qualified Name. This is needed to properly evaluate the namespace prefix, if used.

The qname extension may appear within the type-stmt, within a typedef, leaf, or leaf-list. The builtin data type must be 'string', or the 'qname' extension will be ignored.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
leaf bad-element {
  ncx:qname;
  type string;
}
```

15.5.18 oc-ext:openconfig-hashed-value

The **oc-ext:openconfig-hashed-value** extension is used to indicate that the leaf or leaf-list contains an openconfig hashed password string.. The openconfig-extensions module contains this extension. It is available from the openconfig github repository, and not included in the YumaPro SDK distribution.

The qname extension may appear as a sub-statement of a leaf or leaf-list statement. It will be ignored otherwise.

If this extension is found then the leaf or leaf-list will be treated as an OpenConfig Hashed Value:

- value passed by client is a cleartext password
  - It is not a crypt-hash beginning with the prefix “0”
- server immediately converts the cleartext to a hash
- Only the hash value is ever returned to a client in a data retrieval operation
- Only the hash value is stored in NV-storage (E.g., startup-cfg.xml)

YANG File: **Not Available in YumaPro SDK**

Argument: none

Example:

```
module test-ochash {
  yang-version 1.1;
  namespace "http://netconfcentral.org/ns/test-ochash";
```

```
    prefix toch;
    import openconfig-extensions { prefix oc-ext; }

    revision 2021-02-26;

    container octop {
      leaf ochash {
        type string;
        oc-ext:openconfig-hashed-value;
      }
    }

}
```

Example: Set the ochash leaf to the value ‘this is a test’

Note the value is passed as a string (highlighted below)

```
Incoming msg for session 3
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="2"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
   <candidate/>
  </target>
  <default-operation>merge</default-operation>
  <test-option>set</test-option>
  <config>
   <octop xmlns="http://netconfcentral.org/ns/test-ochash">
    <ochash
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     nc:operation="merge">this is a test</ochash>
   </octop>
  </config>
 </edit-config>
</rpc>
```

If logging is enabled a similar log trace should be present to this example:

```
agt_crypt: altering openconfig-hashed-value value from cleartext to '$6$PE1fq4I22dYjomxn$'
for node 'ochash'
```

If the data is retrieved (E.g., get-config from <candidate>) then the hashed value will be returned.

The server never stores the original plaintext version and it cannot be retrieved from the server at all.

```
<data>
```

```
  <octop xmlns="http://netconfcentral.org/ns/test-ochash">
   <ochash>$6$PE1fq4I22dYjomxn$E60QUJiDIB6uZ.cx7WbsWATrs/.2Zp8NROf7SW3VnSrM4ZrTBlJUHbjtxxMb/
hFcPxgdxZOQdzzKNRWiX6AF2/</ochash>
  </octop>
 </data>
```

15.5.19 oc-ext:regexp-posix

The **oc-ext:regexp-posix** extension is used to indicate that the module uses Posix pattern statements instead of the XSD syntax defined in YANG. The openconfig-extensions module contains this extension. It is available from the openconfig github repository, and not included in the YumaPro SDK distribution.

The qname extension may appear as a body-stmt at the top-level of the module. It will be ignored otherwise.

If this extension is found then the module will be treated as an OpenConfig style module usingt Posix pattern statements.

This extension overrides the **--with-ocpattern** CLI parameter, so even if this parameter is false or the module does not begin with the string “openconfig-”, the Posix pattern syntax will be used.

YANG File: **Not Available in YumaPro SDK**

Argument: none

Example:

```
module my-openconfig-ext {
   ….

   import openconfig-extensions { prefix oc-ext; }

   ….

   oc-ext:regexp-posix;

}
```

15.5.20 ncx:root

The **ncx:root** extension is used within a container definition to indicate it is really a root container for a conceptual NETCONF database, instead of just an empty container. This is needed for YumaPro to correctly process any RPC method that contains a 'config' parameter.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
rpc my-edit {
  input {
    container my-config {
       ncx:root;
    }
  }
}
```

15.5.21 ywx:rpc-root

The yw**x:rpc-root** extension is used internally with the YANG-API protocol to identify a container definition to indicate it is really a root container for a conceptual NETCONF operations, instead of just a container. The container is expected to be empty. Any top-level rpc-stmt can be specified using a QName value with the same module and local name as the RPC operation definition.

This extension is reserved and only used internally by the server.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

15.5.22 ncx:schema-instance

The **ncx:schema-instance** extension is used to indicate that the typedef or type statement for a string data type really identifies a special schema-instance node, not a generic string. A schema-instance value string is an unrestricted YANG instance-identifier expression. All the same rules as an instance-identifier apply except:

- predicates for keys are optional
  The dataRule will apply to all instance of any missing key leaf predicate.

This extension will be ignored unless it is present in the type-stmt of a typedef-stmt, leaf-stmt, or leaf-list-stmt, or directly within a leaf-stmt or leaf-list-stmt..

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
leaf target {
  ncx:schema-instance;
  type string {
    length “1 .. max”;
  }
}
```

15.5.23 nacm:secure

The **nacm:secure** extension is used to indicate that the data model node represents a sensitive security system parameter. It is equivalent to the IETF nacm:default-deny-write extension. The IETF access control model is derived from the yuma-nacm module.

If present, the NETCONF server will only allow the designated 'superuser' to have write or execute default nacm-rights for the node. An explicit access control rule is required for all other users.

The 'secure' extension may appear within a data, rpc, or notification node definition. It is ignored otherwise.

This extension is obsolete and should not be used. Use nacm:default-deny-write instead.

YANG File: **netconf/modules/netconfcentral/yuma-nacm.yang**

Argument: none

Example:

```
leaf mtu {
  nacm:secure;
  type string {
    length “1 .. max”;
  }
}
```

15.5.24 ncx:sil-aio-get2

The **ncx:sil-aio-get2** extension is used within a data definition statement to define the GET2 retrieval mechanism. This extension affects the descendant data nodes.

This extension can be used in a container or list to force the server to treat that data subtree as a All In One AIO node for GET2 callback.

The entire subtree would be expected in one retrieval in one callback invocation.

The entire subtree can be specified in the JSON or XML buffer that will be used for return values. The server will parse and handle the buffer and process retrieval based on the provide JSON or XML encoded buffer. The 'parmstr' argument can specify the encoding that will be used in the callback. Available options are:

- val: val_value_t tree is expected in return value
- xml: XML element in a buffer is expected in return value
- json: JSON object in a buffer is expected in return value

If not specified default val value retrieval mechanism will be assumed.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: string (val | xml | json)

Example:

```
/* All in One callback for Container with List */
container get3-cont-list {            // All in One get2 CB
  ywx:sil-aio-get2 "json";
  config false;

  leaf D { type int8; }

  list nonconfig-list {               // No callback
    key name;
    leaf name { type string; }
    leaf not-keyname { type string; }

    container int-con {               // No callback
      leaf con-leaf { type int32; }
    }
  }
}
```

15.5.25 ncx:sil-delete-children-first

The **ncx:sil-delete-children-first** extension is used within a container or list definition to indicate that the SIL callbacks for descendant nodes should be invoked first, when a data node instance of the object containing this extension is deleted.

Normally, the parent node is expected to delete all its own sub-structures when the SIL edit callback is invoked. If this extension is present, then any SIL callbacks for any of the child nodes will be invoked first instead. If a child node is a list or a container, and it also contains an 'ncx:sil-delete-children-first' extension, then its children will be checked first.

The SIL edit callback will not be invoked for leaf, leaf-list, or anyxml descendant nodes in this mode. They will only will called if their parent node is not getting deleted.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
  container foo {
      ncx:sil-delete-children-first;
      list foos {
        ncx:sil-delete-children-first;
        key a;
        leaf a { type string; }
        container b {
          list c { ... }
        }
        leaf d { type empty; }
      }
    }

  In this example, assume node /foo gets deleted.
  Then the SIL edit callbacks would be done as follows:

  1) /foo/foos[a='n']/b   (called for row 'n' of /foo/foos)
  2) /foo/foos[a='n']     (called for row 'n' of /foo/foos)
  3) repeat (1,2) until all rows in /foo/foos are deleted
  4) /foo

Note that the SIL edit callback is not done for list
/foo/foos[a='n']/b/c because this extension is
not present in container '/foo/foos/b'.

Note that the SIL edit callback is not done for
nodes /foo/foos[a='n']/a or /foo/foos[a='n']/d because
they are leafs.
```

15.5.26 ywx:sil-force-replace-replay

The **ywx:sil-force-replace-replay** extension is used within a configuration data node definition statement to indicate that the SIL (or SIL-SA) callback should be invoked even for nodes that are not changing, during a replace operation. All SIL callbacks for child nodes in the replace request (where the parent node contains this extension) will be invoked during edit processing. This extension can be used with the **sil-force-replay** extension.

If this extension is used within a list statement, then SIL callbacks for all instances of the list that are provided in the replace operation will be invoked.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: none

Example:

```
container top {
  list bar {
    ywx:sil-force-replace-replay;

    key name;
    leaf name { type string; }
    leaf b { type int32; }
    leaf c { type int32; }
  }
}

Replace config:
<config>
  <top nc:operation="replace">
    <bar>   // this entry is changing
      <name>fred</name>
      <b>22</b>
      <c>99</c>
    </bar>
    <bar>   // not changing but SIL will be called anyway
      <name>barney</name>
      <b>18</b>
      <c>82</c>
    </bar>
  </top>
</config>


In this example, all instances of /top/bar will get replaced.
Normally the server will skip replacement of instances which
are not changing at all.  If this extension is present, then
the server will not skip any instances that are provided.
Instead, the SIL edit callbacks would be done for these
nodes as well. The 'newval' and 'curval' parameters will
be the same for the replayed entries.
```

15.5.27 ywx:sil-force-replay

The **ywx:sil-force-replay** extension is used within a container or list definition to indicate that the SIL callbacks for all child nodes should be invoked when one of the child nodes is modified.

Normally, only the SIL callbacks for the child nodes that are changed are called during an edit transaction.

If the parent node for the changed child node contains the “sil-force-replay” extension, then all the child node SIL callbacks will be invoked instead.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: none

Example:

```
list foo {
    ywx:sil-force-replay;
    key a;
    leaf a { type string; }
    container b {
      list c { ... }
    }
    leaf d { type int32; }
    leaf e { type int32; }
}

In this example, assume node /foo/d gets modified.
Then the SIL edit callbacks would be done for sibling nodes as well.
The 'newval' and 'curval' parameters will be the same for the replayed
sibling nodes.

1) /foo/a
2) /foo/b
3) /foo/d
4) /foo/e
```

15.5.28 ywx:sil-priority

The **ywx:sil-priority** extension is used within a configuration data node definition to set the SIL priority for an object. The lower the number of the SIL priority, the higher priority it is assigned. SIL callbacks are normally invoked in the order that the edits appear in the edit request. If the sil-priority is set then the order SIL callbacks are invoked will be based on the numeric priority value instead.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: prio: number from 1 to 255

Example:

```
"Used to control the order that SIL or SIL-SA callbacks
 are invoked for specific objects.

 If this extension is used within a configuration database
 object then the SIL priority for the object will be assigned
 the value of the 'prio' argument.

 Only the order of the 'apply', 'commit' and 'rollback'
 callback phases will be affected by this parameter.
 The 'validate' phase callbacks are invoked in the
 order they appear in the edit request.

 The 'prio' argument must be a number between 1 and 255.
 If two objects are edited in the same edit request, the
 the one with the lowest SIL priority number will be
 executed first.

 If no sil-priority is set, then the default value of
 '255' will be used instead.

 If the SIL priority is the same for two objects in the
 same edit request, then the server will pick an order
 in an implementation-specific manner.";

 leaf A1 {
   type string;
    ywx:sil-priority 30;
 }

 leaf A2 {
   type string;
   ywx:sil-priority 20;
 }

 leaf A3 {
  type string;
  ywx:sil-priority 10;
}
```

15.5.29 ywx:sil-test-get-when

The **ywx:sil-test-get-when** extension is used within an operational data node definition to set thesil-get-test-when parameter setting for an object. This parameter controls whether “when-stmt” expressions will be evaluated by the server. Normally the SIL or SIL-SA code is expected to check these conditions but the server can run the Xpath test during retrieval operations.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: boolval: boolean (true or false)

```
extension sil-test-get-when {
  description
    "Used within a data definition statement to define
     the --sil-get-test-when CLI parameter behavior for
     a single object. This extension does not affect the
     descendant data nodes.

     The 'boolval' argument must be the string 'true'
     or 'false'; If 'true' the object will be tested
     for when-stmts if any need to be evaluated during
     retrieval operations. If 'false' then any when-stmts
     will be ignored during retrieval operations.

     This extension will override the --sil-test-get-when
     global CLI parameter. This extension will have no affect
     unless the value is different than this CLI parameter.
   ";
   argument boolval;
 }
```

Example:

```
leaf myoperleaf {
  ywx:sil-test-get-when true;
  type string;
}
```

15.5.30 ywx:urlpath

The **ywx:urlpath** extension is used to indicate that a string data node is really using the YANG-API URL path expression syntax. It can be used within a leaf or leaf-list definition.

YANG File: **netconf/modules/yumaworks/yumaworks-extensions.yang**

Argument: none

Example:

```
leaf restpath {
  ywx:urlpath;
  type string;
}

<restpath>/interfaces/interface/eth0/mtu</restpath>
```

15.5.31 ncx:user-write

The **ncx:user-write** extension is used within database configuration data definition statements to control user write access to the database object containing this statement.

The 'exceptions' argument is a list of operations that users are permitted to invoke for the specified node. These permissions will over-ride all NACM access control rules, even if NACM is disabled.

- This extension does not apply to descendant nodes!
- This extension has no effect if config-stmt is false!

The following values are supported:

- create : allow users to create instances of the object
- update : allow users to modify instances of the object
- delete : allow users to delete instances of the object

To dis-allow all user access, provide an empty string for the 'exceptions' argument (user-write '';)

To allow only create and delete user access, provide the string 'create delete' for the 'exceptions' parameter.

Use this for parameters that cannot be changed once they are set. Providing all 3 parameters has the same affect as not using this extension at all, but can be used anyway.

YANG leaf version:

```
    leaf user-write {
        type bits {
          bit create;
          bit update;
          bit delete;
        }
        default 'create update delete';
        description 'equivalent YANG definition';
      }
```

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument:

- exceptions: the list of operations to permit for the data node. The YANG bits definition shown above defines the syntax for the exceptions string.

Example:

```
container interfaces {
  ncx:user-write “update”;
  list physical-interface {
    ncx:user-write “update”;
    …
  }
}
```

15.5.32 nacm:very-secure

The **nacm:very-secure** extension is used to indicate that the data model node represents a sensitive security system parameter. It is equivalent to the IETF nacm:default-deny-all extension. The IETF access control model is derived from the yuma-nacm module.

If present, the NETCONF server will only allow the designated 'superuser' to have read, write, or execute nacm-rights for the node. An explicit access control rule is required for all other users.

The 'very-secure' extension may appear within a data, rpc, or notification node definition. It is ignored otherwise.

This extension is obsolete and should not be used. Use nacm:default-deny-all instead.

YANG File: **netconf/modules/netconfcentral/yuma-nacm.yang**

Argument: none

Example:

```
leaf social-security-id {
  nacm:very-secure;
  type string {
    length “1 .. max”;
  }
}
```

15.5.33 ncx:xpath

The **ncx:xpath** extension is used to indicate that the content of a data type is an XPath expression. This is needed to properly evaluate the namespace prefixes within the expression.

Note that this extension is deprecated. Use the xpath1.0 data type found in ietf-yang-types.yang

The xpath extension may appear within the type-stmt, within a typedef, leaf, or leaf-list. The builtin data type must be 'string', or the 'xpath' extension will be ignored.

All data using the 'instance-identifier' built-in type will automatically be processed as an XPath string, so the xpath extension is not needed in that case.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument: none

Example:

```
leaf target {
  ncx:xpath;
  type string {
    length “1 .. max”;
  }
}
```

15.5.34 ywx:xpath-operational-ok

The **ywx:xpath-operational-ok** extension is used to indicate that the “must” and “when” expressions within the indicated object are allowed to access config=false data nodes.

This extension is used within a data-definition statement for a configuration data node to alter the must-stmt and when-stmt found within the data node. This allows an XPath expression in such a node to reference config=false data nodes.

This property does not apply to any child nodes, just the data node containing this external statement.

This violates the standard in RFC 7950, sec 6.4.1 so use with caution since the YANG module will not be valid according to YANG 1.1 rules.

There is no parameter for this extension.

In the following example, the list “/c3/l3” has a when-stmt that accesses the /interrfaces-state subtree:

```
module mytest2 {
  namespace "urn:yumaworks:params:xml:ns:yang:mytest2";
  prefix "mt2";
  import ietf-interfaces { prefix if; }
  import yumaworks-extensions { prefix ywx; }

  revision 2019-12-06 {
  description
  "Initial version";
  }

  container c3 {
    list l3 {
      ywx:xpath-operational-ok;
      when "/if:interfaces-state/if:interface/if:speed > 1000";
      key "name";
      leaf name{type string;}
      leaf if-index {type int32;}
    }
  }
}
```

15.5.35 ncx:xsdlist

The **ncx:xsdlist** extension is used to indicate the leaf string type is really an XSD list, which is a series of white space separated token strings. The type argument represents the data type to use for the list members, for validation purposes. This extension is allowed to be present within the type sub-section for a string.

YANG File: **netconf/modules/netconfcentral/yuma-ncx.yang**

Argument:

- **type:** name of the data type to use for list members

Example:

```
typedef MyEnumType {
  enum up;
  enum down;
  enum left;
  enum right;
}

leaf enum-list {
  ncx:xsdlist MyEnumType;
  type string;
}

<enum-list>up right down</enum-list>
```