YumaPro Server
yp-system API Guide

YANG-Based Unified Modular Automation Tools

Server Instrumentation Library Development

Version 19.10-12

Table of Contents

1 Preface

1.1 Legal Statements

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

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

1.2 Additional Resources

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

YumaPro Installation Guide

YumaPro Quickstart Guide

Other documentation includes:

YumaPro API Quickstart Guide

YumaPro User Manual

YumaPro netconfd-pro Manual

YumaPro yangcli-pro Manual

YumaPro yangdiff-pro Manual

YumaPro yangdump-pro Manual

YumaPro Developer Manual

YumaPro ypclient-pro Manual

YumaPro yp-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 Introduction

This document contains a quick reference for the API callback interface for the **netconfd-pro** server.

These interfaces are explained in detail in the YumaPro Server Developer Manual.

There are 6 sections:

- **yp-system:** Creating and using the general system SIL library
- **RPC operations:** Validation and invocation callbacks for protocol operations
- **Operational data:** Retrieval callbacks for operational data
- **Configuration data:** Editing callbacks for configuration datastores
- **Notifications:** NETCONF notification callbacks for event generation
- **Error handling:** API functions for <rpc-error> generation

3 yp-system External Interface

The **netconfd-pro** server supports an external vendor library that is loaded at boot-time by the server, similar to a SIL for a YANG module. This library code is run in the context of the main server process. It is used to hook vendor-specific functions into the server,.

The following tasks are supported by the yp-system library interface:

- initialization
- cleanup
- hook in external access control model
- hook in external syslog interface

The default system library is located in called **libyp_system.so**, it it is located in the library path, such as **/usr/lib/yumapro/libyp_system.so.**

There is an example **yp-system** library in the **libsystem** directory of the YumaPro source tree. This file in the 'src' directory called **example-system.c** contains some stub code showing how this library is used.

3.1 Mandatory yp-system API Functions

The following API functions should be defined in all yp-library code:

- **yp_system_init_profile**: Change default settings in the server profile defined in agt/agt.h
- **yp_system_init1**: Phase 1 initialization function
- **yp_system_init2**: Phase 2 initialization function
- **yp_system_cleanup**: Server cleanup function

The following example shows **example-system.h** contents from the **libsystem/src** directory:

```
#ifndef _H_example_system
#define _H_example_system
/*
 * Copyright (c) 2012, YumaWorks, Inc., All Rights Reserved.
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

#ifndef _H_agt
#include "agt.h"
#endif

#ifndef _H_status
#include "status.h"
#endif


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


/* 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
 */
```

```
extern status_t yp_system_init1 (boolean pre_cli);


/* 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
 */
extern status_t yp_system_init2 (boolean pre_load);


/* system cleanup callback
 * this callback is invoked once during agt_cleanup
 */
extern void yp_system_cleanup (void);


#endif
```

3.2 YP-HA Interface Functions

The High Availability features in the netconfd-pro server can be accessed from the yp-system library.

There are two functions that call be called from the “init1” or “init2” phases in the yp-system library initialization. They can also be called from an agt_timer callback function during system runtime.

Normally when the server running with YP-HA enabled (--ha-enabled=true), the server will wait until it is told which HA role should be used (active or standby). The –ha-initial-active CLI parameter can be used to hard-wire the YA role in the configuration. This should only be used for debugging. In production networks, the HA management system should signal all the HA role changes to the server.

The YP-HA role can be set from internal API functions of from the DB-API subsystem interface, using the “yp-ha-role” message.

3.2.1 agt_ha_role_t

The **agt_ha_role_t** is an enumeration specifying the different YP-HA roles.

```
/* server HA mode mode */
typedef enum agt_ha_role_t_ {
    AGT_HA_ROLE_NONE,
    AGT_HA_ROLE_NOT_SET,
    AGT_HA_ROLE_STANDBY,
    AGT_HA_ROLE_ACTIVE
} agt_ha_role_t;
```

3.2.2 agt_yp_ha_get_role

The **agt_yp_ha_get_role()** function is used to retrieve the current YP-HA role used by the server.

```
/********************************************************************
* FUNCTION agt_yp_ha_get_role
*
* Get the YP-HA server role for this server
* RETURNS:
*   server role enum
*********************************************************************/
extern agt_ha_role_t
    agt_yp_ha_get_role (void);
```

3.2.3 agt_yp_ha_be_active

The **agt_yp_ha_be_active** function is used to tell the server to be the active server in the YP-HA protocol.

```
/*********************************************************************
* FUNCTION agt_yp_ha_be_active
*
* Put this server in YP-HA Active mode
* RETURNS:
*
*********************************************************************/
extern status_t
    agt_yp_ha_be_active (void);
```

3.2.4 agt_yp_ha_be_standby

The **agt_yp_ha_be_standby** function is used to tell the server to be a standby server in the YP-HA protocol.

```
/*********************************************************************
* FUNCTION agt_yp_ha_be_standby
*
* Put this server in YP-HA Standby mode
* RETURNS:
*   status == NO_ERR if this server able to enter YP-HA Standby mode
*********************************************************************/
extern status_t
    agt_yp_ha_be_standby (const xmlChar *new_server_id);
```

3.2.5 agt_yp_ha_be_none

The **agt_yp_ha_be_none** function is used to tell the server to leave its current YP-HA role, go offline, and wait for a new YP-HA role to be set.

```
/*********************************************************************
* FUNCTION agt_yp_ha_be_none
*
* Put this server in WAIT_ROLE state
* RETURNS:
*   status
*********************************************************************/
extern status_t
    agt_yp_ha_be_none (void);
```

3.2.6 DB-API <yp-ha-role> Event Message

The DB-API service has a new utility function to set the YP-HA role on the local server.

The <yp-ha-mode> message is sent from a subsystem to the main server. It has 2 variants. Either the "go-active" or the "go-standby" variant is sent.

```
        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";
            }
            container go-standby {
              description "Become a YP-HA standby server";
              leaf new-active {
                type string;
                mandatory true;
                description
                  "Server name of the active server to use";
              }
            }
          }
        }  // container yp-ha-mode
```

Example message to set the YP-HA Active mode:

```
<?xml version="1.0" encoding="UTF-8"?>
<ycontrol xmlns="http://yumaworks.com/ns/yumaworks-ycontrol">
  <message-id>2</message-id>
  <message-type>subsys-event</message-type>
  <server-id>ha-1</server-id>
  <subsys-id>subsys1</subsys-id>
  <service-id>db-api</service-id>
  <payload>
    <db-api xmlns="http://yumaworks.com/ns/yumaworks-db-api">
       <yp-ha-mode>
          <go-active/>
      </yp-ha-mode>
    </db-api>
  </payload>
</ycontrol>
```

Example message to set the YP-HA Standby mode:

```
<?xml version="1.0" encoding="UTF-8"?>
<ycontrol xmlns="http://yumaworks.com/ns/yumaworks-ycontrol">
  <message-id>2</message-id>
  <message-type>subsys-event</message-type>
  <server-id>ha-1</server-id>
  <subsys-id>subsys1</subsys-id>
  <service-id>db-api</service-id>
  <payload>
    <db-api xmlns="http://yumaworks.com/ns/yumaworks-db-api">
       <yp-ha-mode>
          <go-standby>ha-2</go-standby>
      </yp-ha-mode>
    </db-api>
  </payload>
</ycontrol>
```

4 Access Control Model Interface

The **netconfd-pro** server supports 3 different access control models:

- **ietf**: IETF Proposed Standard YANG data model (ietf-netconf-acm.yang) in RFC 6536 is used
- **yuma**: original Netconf Central NACM module (yuma-nacm.yang) is used
- **external**: an external data model provided by the vendor is used

If the **external** mode is used, then ACM callback functions must be registered with API functions that the server needs to validate the following types of access:

- protocol operation requests
- database read and write requests
- notification delivery

4.1 Mandatory External ACM Callback Functions

The function **agt_acm_extern_register_callbacks** in **agt/agt_acm_extern.h** is used by the external system code to register its own callback functions to handle access control requests. These functions are used by the system instead of its own access control cache and data model.

If the external access control method is selected in the system initialization then these callback functions MUST be provided or all access requests will be granted by default!

The following functions must be registered:

- **RPC Request Fn**: Checks if user is allowed to invoke the specified YANG-defined rpc operation. The **agt_acm_extern_rpc_fn_t** template is used for this callback.
- **Notification Send Fn**: Checks if user is allowed to receive the specified YANG-defined notification event. The **agt_acm_extern_notif_fn_t** template is used for this callback.
- **Database Write Request Fn**: Checks if user is allowed to edit the specified YANG-defined data node. The **agt_acm_extern_write_fn_t** template is used for this callback.
- **Database Read Request Fn**: Checks if user is allowed to read the specified YANG-defined data node. The **agt_acm_extern_read_fn_t** template is used for this callback.

The following code snippet shows the API template definitions from agt/agt_acm_extern.h:

```
/********************************************************************
* FUNCTION agt_acm_extern_rpc_fn
*
* Check if the specified user is allowed to invoke an RPC
*
* INPUTS:
*   msg == XML header in incoming message in progress
*   user == user name string
*   rpcobj == obj_template_t for the RPC method to check
*
* RETURNS:
*   TRUE if user allowed invoke this RPC; FALSE otherwise
********************************************************************/
typedef boolean
    (*agt_acm_extern_rpc_fn_t) (xml_msg_hdr_t *msg,
                                const xmlChar *user,
                                const obj_template_t *rpcobj);


/********************************************************************
* FUNCTION agt_acm_extern_notif_fn
*
* Check if the specified user is allowed to receive
* a notification event
*
* INPUTS:
*   user == user name string
*   notifobj == obj_template_t for the notification event to check
*
```

```
* RETURNS:
*   TRUE if user allowed receive this notification event;
*   FALSE otherwise
*********************************************************************/
typedef boolean
    (*agt_acm_extern_notif_fn_t) (const xmlChar *user,
                                  const obj_template_t *notifobj);


/*********************************************************************
* FUNCTION agt_acm_extern_write_fn
*
* Check if the specified user is allowed to access a value node
* The val->obj template will be checked against the val->editop
* requested access and the user's configured max-access
*
* INPUTS:
*   msg == XML header from incoming message in progress
*   newval  == val_value_t in progress to check
*               (may be NULL, if curval set)
*   curval  == val_value_t in progress to check
*               (may be NULL, if newval set)
*   val  == val_value_t in progress to check
*   editop == requested CRUD operation
*
* RETURNS:
*   TRUE if user allowed this level of access to the value node
*********************************************************************/
typedef boolean
    (*agt_acm_extern_write_fn_t) (xml_msg_hdr_t *msg,
                                  const xmlChar *user,
                                  const val_value_t *newval,
                                  const val_value_t *curval,
                                  op_editop_t editop);


/*********************************************************************
* FUNCTION agt_acm_extern_read_fn
*
* Check if the specified user is allowed to read a value node
*
* INPUTS:
*   msg == XML header from incoming message in progress
*   user == user name string
*   val  == val_value_t in progress to check
*
* RETURNS:
*   TRUE if user allowed read access to the value node
*********************************************************************/
typedef boolean
    (*agt_acm_extern_read_fn_t) (xml_msg_hdr_t *msg,
                                 const xmlChar *user,
                                 const val_value_t *val);
```

4.2 Example External ACM

The following example code is available in **example-system.c,** found in the l**ibsystem/src** directory. It shows some dummy external ACM functions and how they are registered during initialization.

```
/*
 * Copyright (c) 2012, YumaWorks, Inc., All Rights Reserved.
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
/*  FILE: example-system.c

  Example External System Library

*********************************************************************
*                                                                   *
*                     I N C L U D E    F I L E S                    *
*                                                                   *
*********************************************************************/

#include <xmlstring.h>

#include "procdefs.h"
#include "agt.h"
#include "agt_acm.h"
#include "agt_acm_extern.h"
#include "agt_util.h"
#include "dlq.h"
#include "example-system.h"
#include "log.h"
#include "ncx.h"
#include "ncxtypes.h"
#include "obj.h"
#include "ses.h"
#include "status.h"
#include "val.h"
#include "val_util.h"
#include "xml_util.h"



/***************  Example External NACM Hooks ******************/


/*********************************************************************
* FUNCTION acm_extern_rpc
*
* Check if the specified user is allowed to invoke an RPC
*
* INPUTS:
*   msg == XML header in incoming message in progress
*   user == user name string
*   rpcobj == obj_template_t for the RPC method to check
```

```
*
* RETURNS:
*   TRUE if user allowed invoke this RPC; FALSE otherwise
*********************************************************************/
static boolean
    acm_extern_rpc (xml_msg_hdr_t *msg,
                    const xmlChar *user,
                    const obj_template_t *rpcobj)
{
    (void)msg;
    (void)user;
    (void)rpcobj;
    log_debug("\nacm_extern_rpc: return OK\n");
    return TRUE;
}


/*********************************************************************
* FUNCTION acm_extern_notif
*
* Check if the specified user is allowed to receive
* a notification event
*
* INPUTS:
*   user == user name string
*   notifobj == obj_template_t for the notification event to check
*
* RETURNS:
*   TRUE if user allowed receive this notification event;
*   FALSE otherwise
*********************************************************************/
static boolean
    acm_extern_notif (const xmlChar *user,
                      const obj_template_t *notifobj)
{
    (void)user;
    (void)notifobj;
    log_debug("\nacm_extern_notif: return OK\n");
    return TRUE;
}


/*********************************************************************
* FUNCTION acm_extern_write_fn
*
* Check if the specified user is allowed to access a value node
* The val->obj template will be checked against the val->editop
* requested access and the user's configured max-access
*
* INPUTS:
*   msg == XML header from incoming message in progress
*   newval  == val_value_t in progress to check
*                (may be NULL, if curval set)
*   curval  == val_value_t in progress to check
*                (may be NULL, if newval set)
*   val  == val_value_t in progress to check
*   editop == requested CRUD operation
*
* RETURNS:
*   TRUE if user allowed this level of access to the value node
*********************************************************************/
static boolean
    acm_extern_write (xml_msg_hdr_t *msg,
```

```
                      const xmlChar *user,
                      const val_value_t *newval,
                      const val_value_t *curval,
                      op_editop_t editop)
{
    (void)msg;
    (void)user;
    (void)newval;
    (void)curval;
    (void)editop;
    log_debug("\nacm_extern_write: return OK\n");
    return TRUE;
}


/*********************************************************************
* FUNCTION acm_extern_read_fn
*
* Check if the specified user is allowed to read a value node
*
* INPUTS:
*   msg == XML header from incoming message in progress
*   user == user name string
*   val  == val_value_t in progress to check
*
* RETURNS:
*   TRUE if user allowed read access to the value node
*********************************************************************/
static boolean
    acm_extern_read (xml_msg_hdr_t *msg,
                     const xmlChar *user,
                     const val_value_t *val)
{
    (void)msg;
    (void)user;
    (void)val;
    log_debug("\nacm_extern_read: return OK\n");
    return TRUE;
}



/****************  Required System Library Hooks ******************/

/* system init server profile callback
 *
 * Initialize the server profile if needed
 *
 * INPUTS:
 *  profile == server profile to change if needed
 */
void yp_system_init_profile (agt_profile_t *profile)
{
    log_debug("\nyp_system init profile\n");
    /* example: use an external ACM module */
      profile->agt_acm_model = AGT_ACM_MODEL_EXTERNAL;
}


/* 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
 */
status_t yp_system_init1 (boolean pre_cli)
{
    log_debug("\nyp_system init1\n");

    if (pre_cli) {
        ;
    } else {
        // example -- external NACM callbacks
        // load module for external module
        // with ncxmod_load_module

        // register the external ACM callbacks
        // this will have no affect unless the
        // yp_system_init_profile fn sets the
        // agt_acm_model to AGT_ACM_MODEL_EXTERNAL
        agt_acm_extern_register_callbacks(acm_extern_rpc,
                                          acm_extern_notif,
                                          acm_extern_write,
                                          acm_extern_read);
    }
    return NO_ERR;
}


/* 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
 */

status_t yp_system_init2 (boolean pre_load)
{
    log_debug("\nyp_system init2\n");

    if (pre_load) {
        ;
    } else {
        ;
    }
    return NO_ERR;
}


/* system cleanup callback
 * this callback is invoked once during agt_cleanup
 */
void yp_system_cleanup (void)
{
```

```
    log_debug("\nyp_system cleanup\n");
}

/* END example-system.c */
```

5 Syslog Interface

The external syslog interface allows a vendor-specific syslog send function to be used instead of the Linux system syslog daemon. The **yp_library** user library code can be used to initialize an external syslog send function.

The external syslog send function must follow the C function templates defined in **netconf/src/ncx/log_vendor_extern.h**:

- **logfn_vendor_init1_t:** Phase 1 initialization function
- **logfn_vendor_send_t:** syslog send function
- **logfn_vendor_cleanup_t:** Termination cleanup function

The 'example-system C and H files in the yp-system directory contains example usage of these callback functions.

When the vendor callback is invoked, the vendor code should translate the YumaPro application parameters into an "application" and/or "facility" equivalent appropriate to the vendor logging schema. Likewise, it should translate the YumaPro level parameters into a "message type/level" appropriate to its own requirements.

```
/* external logger init function */
typedef void (*logfn_vendor_init1_t) ( boolean pre_cli );

/* external logger send function */
typedef void (*logfn_vendor_send_t) ( log_debug_app_t app, log_debug_t level,
                                      const char *fstr, va_list args );

/* external logger cleanup function */
typedef void (*logfn_vendor_cleanup_t) ( void );
```

6 yp-system Library

The **yp-system** library contains special SIL code for global APIs that are not specific to 1 YANG module.

The **libsystem/src** directory contains the file **example-system.c** and **example-system.h.** These contain empty callbacks that can be filled in.

The mandatory to implement callback functions for server initialization and cleanup are described in this section.

6.1 Copy and Setup libsystem subtree

The default library created by building **libsystem** is **libyp_system-example.so.** This is not the correct filename, in order to prevent the real **yp-server** library from being overwritten by the empty example library, if “make install” is run.

The **libsystem** subtree should be copied to your own development area and modified.

Instructions are in the src/Makefile:

```
#
# to make a real library, copy this directory contents
# to a new location and change yp-system-example to yp_system
# in the SUBDIR_NM macro below
#
#  SUBDIR_NM=yp_system
#
SUBDIR_NM=yp_system-example
```

Change the string **yp-system-example** to **yp-system** in the src/Makefile in the copy of **libsystem**.

6.2 yp_system_init_profile

This function is used to modify the defaults in the **agt_profile_t** structure used by the server. This allows features and various behavior to be configured. Many of these settings have associated CLI/conf parameters that can change the default at run-time.

```
/* system init server profile callback
 *
 * Initialize the server profile if needed
 *
 * INPUTS:
 *  profile == server profile to change if needed
 *
 * OUTPUTS:
 *   *profile fields can be altered as needed to change defaults
 */
extern void yp_system_init_profile (agt_profile_t *profile);
```

The contents of this function should set profile fields as needed.

Note that if the field has a CLI override parameter with a default value , then the YANG default for that parameter needs to be changed if the profile default is changed.

Example Profile Initialization Function

This example profile initialization function shows how to disable all the Yuma YANG modules:

```
/********************************************************************
* FUNCTION yp_system_init_profile
*
* Set vendor configured server profile parameters.
*
* OUTPUTS:
*  *profile is filled in with params or defaults
*
*********************************************************************/
void yp_system_init_profile (agt_profile_t *profile);
{

    profile->agt_use_yuma_arp = false;
    profile->agt_use_yuma_if = false;
    profile->agt_use_yuma_mysession = false;
    profile->agt_use_yuma_proc = false;
    profile->agt_use_yuma_proc = false;

} /* yp_system_init_profile */
```

6.2.1 Datastore Fields

agt_autodelete_pdu_error

Description	The **agt_autodelete_pdu_erro** field specifies whether configuration nodes provided in the edit payload (e.g., <config> element) that are conditional on 'when' statements must evaluate to true or else an operation-failed error will be returned.
Value	(boolean) • **false**: then such 'false when' will be silently removed from the target datastore.
CLI Override	--**autodelete-pdu-error** parameter
Default	true

agt_create_empty_npcontainers

Description	An empty non-presence container has no meaning in NETCONF/YANG so it may be created by the server. In particular, the presence of the container node with no child nodes is semantically equivalent to the absence of the container node. This is the default style.
Value	(boolean) • false: the server will not create empty NP containers.
CLI Override	--**create-empty-npcontainers** parameter
Default	true

agt_backup_dir

Description	The **agt_backup_dir** field specifies the directory location the server should use for saving configuration backups.
Value	(string) • directory spec, must be writable by the server. This will enable the <backup> and <restore> operations • NULL: This will disable the <backup> and <restore> operations
CLI Override	None
Default	$HOME/.yumapro/backups

agt_defaultStyle

Description	The **agt_defaultStyle** field specifies how default data nodes will be treated in configuration datastores. Must match the agt_defaultStyleEnum field.
Value	(string) • explicit: explicit default mode

Description	The **agt_defaultStyle** field specifies how default data nodes will be treated in configuration datastores. Must match the agt_defaultStyleEnum field.
	• trim: trim default mode
CLI Override	--**default-style** parameter
Default	explicit

agt_defaultStyleEnum

Description	The **agt_defaultStyleEnum** field specifies how default data nodes will be treated in configuration datastores. Must match the agt_defaultStyle field.
Value	(ncx_withdefaults_t) • NCX_WITHDEF_EXPLICIT: explicit default mode • NCX_WITHDEF_TRIM: trim default mode
CLI Override	--**default-style** parameter
Default	NCX_WITHDEF_EXPLICIT

agt_no_nvstore

Description	The **agt_no_nvstore** field specifies whether the server should not load or save using the normal APIs during transaction management. The 'start' choice will be ignored (e.g., --no-startup) and the server will not attempt to load a startup-cfg.xml file. Transactions will not be saved to NV-storage at all. Any external NV-storage callbacks will be ignored. Use this mode if NV-load and NV-storage are handled internally and not via the startup-cfg.xml file.
Value	empty
CLI Override	--**no-nvstore** parameter
Default	The default is not present.

agt_running_error

Description	The **agt_running_error** field specifies whether the running configuration loaded at boot-time is allowed to have errors in it or not. Therese are validation errors that remain after any error nodes were removed.
Value	(boolean) • true: Allow the running datastore to contain errors that cannot be pruned from the datastore. • false: Exit the program if the startup config has any errors in it.
CLI Override	--**running-error** parameter
Default	false

agt_save_config_system

Description	The **agt_save_config_system** field specifies whether the serverr will skip generation of the startup XML file when a save_config is done by the server. Used with the external config mode where the external system database is already up to date so the XML file is not used
Value	(boolean) • **true**: Skip the generation of the startup XML file when save_config is called to save the startup configuration. • **false**: Do not skip the generation of the startup XML file when save_config is called to save the startup configuration.
CLI Override	none
Default	false

agt_sil_validate_candidate

Description	The **agt_sil_validate_candidate** field specifies whether the server will invoke the VALIDATE phase for SIL and SIL-SA callbacks when each edit is made to the candidate datastore. Transaction performance will be improved if the extra VALIDATE phase callbacks are skipped. Acceptance of an individual edit to the candidate does not mean the SIL or SIL-SA will accept that edit when combined with all edits (during the commit operation). In either case the server will invoke the VALIDATE phase callbacks when an attempt to commit the candidate datastore is done or when a <validate> operation is done on the candidate datastore.
Value	(boolean) • **false**: Do not invoke the VALIDATE phase for SIL and SIL-SA callbacks when each edit is made to the candidate datastore.
CLI Override	--**sil-validate-candidate** parameter
Default	true

agt_sil_skip_load

Description	The **agt_sil_skip_load** field specifies whether the OP_EDITOP_LOAD SIL callbacks will be skipped during server initialization. This is used by programs monitoring the server for crashes, then restarting without resetting the underlying instrumentation.
Value	(boolean) • **true**: Skip the SIL callbacks during the initial load of the startup configuration file into the running datastore • **false**: Do not skip the SIL callbacks during the initial load of the startup configuration file into the running datastore
CLI Override	--**sil-skip-load** parameter
Default	false

agt_start

Description	The **agt_start** field specifies the type of NV-storage that will be used by default.
Value	(ncx_agtstart_t) • **NCX_AGT_START_MIRROR**: Automatically keep the running datastore and non-volatile storage synchronized. • **NCX_AGT_START_DISTINCT**: Support the **:startup** capability. Require the user to manually save the running datastore to non-volatile storage.
CLI Override	--**with-startup** parameter
Default	NCX_AGT_START_MIRROR

agt_startup_error

Description	The **agt_startup_error** field specifies whether the startup configuration loaded at boot-time is allowed to have errors in it or not
Value	(boolean) • **true**: Allow the startup to contain errors that can be pruned from the datastore. • **false**: Exit the program if the startup config has any errors in it.
CLI Override	--**startup-error** parameter
Default	false

agt_system_sorted

Description	The **agt_system_sorted** field specifies whether the ordered-by system YANG data nodes will be sorted or not.
Value	(boolean) • **true**: Keep the system data nodes in sorted order. • **false**: Do not keep the system data nodes in sorted order.
CLI Override	--**system-sorted** parameter
Default	true

agt_targ

Description	The **agt_targ** field specifies the target datastore that will be used by default
Value	(ncx_agttarg_t) • **NCX_AGT_TARG_CANDIDATE**: Support the **:candidate** capability by default. Use the candidate datastore as the default target datastore. • **NCX_AGT_TARG_RUNNING**: Support the **:writable-running** capability. Use the running datastore as the default target datastore

Description	The **agt_targ** field specifies the target datastore that will be used by default
CLI Override	**--target** parameter
Default	NCX_AGT_TARG_CANDIDATE

6.2.2 Logging Fields

agt_audit_log_candidate

Description	The **agt_audit_log_candidate** field specifies if the server will record transactions on the candidate datastore to the audit log.
Value	boolean
CLI Override	--**audit-log-candidate** parameter
Default	true

agt_audit_log_console_level

Description	The **agt_audit_log_console_level** field sets the minimum logging level needed to log datastore audit records to the server console log. This does not affect output to the audit log.
Value	(log_debug_t) enumeration
CLI Override	--**audit-log-console-level** parameter
Default	LOG_DEBUG_DEBUG

agt_audit_log_level

Description	The **agt_audit_log_console_level** field sets the minimum logging level needed to log datastore audit records to the audit log. This does not affect debug logging to the server console log.
Value	(log_debug_t) enumeration
CLI Override	--**audit-log-level** parameter
Default	LOG_DEBUG_INFO

agt_logfile

Description	The **agt_logfile** field specifies the file specification for logging output
Value	string
CLI Override	--**log** parameter
Default	None

agt_logappend

Description	The **agt_logappend** field specifies whether the log file will be appended if found
Value	boolean
CLI Override	--**log-append** parameter
Default	None

agt_log_acm_reads

Description	The **agt_log_acm_reads** field specifies if logging should be enabled for NACM access rule processing. during read operations
Value	boolean
CLI Override	None
Default	false

agt_log_acm_writes

Description	The **agt_log_acm_writes** field specifies if logging should be enabled for NACM access rule processing. during write operations
Value	boolean
CLI Override	None
Default	true

agt_log_level

Description	The **agt_log_level** field specifies the default logging level.
Value	(log_debug_t) enumeration
CLI Override	**--log-level** parameter
Default	LOG_DEBUG_INFO

agt_pthread_log_level

Description	The **agt_pthread_log_level** field specifies the default logging level for PTHREADS debugging.
Value	(log_debug_t) enumeration
CLI Override	**--log-pthread-level** parameter
Default	LOG_DEBUG_INFO

agt_syslog_log_level

Description	The **agt_syslog_log_level** field specifies the default logging level for SYSLOG output.
Value	(log_debug_t) enumeration
CLI Override	**--log-syslog-level** parameter
Default	LOG_DEBUG_INFO

6.2.3 Notification Fields

agt_eventlog_size

Description	The **agt_eventlog_size** field specifies the default size of the replay buffer for NETCONF notifications.
Value	(uint32)
CLI Override	--**eventlog-size** parameter
Default	1000

agt_maxburst

Description	The **agt_maxburst** field specifies the manimum number of notifications to send to a session at once
Value	(uint32) • 0 = no limit • 1 .. N = notification limit
CLI Override	--**max-burst** parameter
Default	10

agt_notif_sequence_id

Description	The **agt_notif_sequence_id** field specifies if the incrementing <sequence-id> element will be added to <notification> messages.
Value	(boolean)
CLI Override	None
Default	false

agt_ietf_system_notifs

Description	The **agt_ietf_system_notifs** field specifies if the **ietf-netconf-notifications** module should be used for system events
Value	(boolean)
CLI Override	--**system-notifications** parameter
Default	true

agt_use_notifications

Description	The **agt_use_notifications** field specifies whether the server should support the **:notifications** and **:interleave** capabilities. If not, then no notifications will be generated, no notification replay buffer processing will be done, and <create-subscription> will not be enabled.
Value	(boolean)
CLI Override	--**with-notifications** parameter
Default	true

agt_yuma_system_notifs

Description	The **agt_yuma_system_notifs** field specifies if the **yuma-system** module should be used for system events.
Value	(boolean)
CLI Override	--**system-notifications** parameter
Default	false

agt_log_event_drops

Description	The **agt_log_event_drops** field specifies whether the server should generate a log entry if dropped because notifications are disabled. Note that drops due to lack of resources are still logged, and not affected by this parameter.
Value	(boolean)
CLI Override	--**log-event-drops** parameter
Default	false

6.2.4 Access Control Fields

agt_accesscontrol

Description	The **agt_accesscontrol** field specifies the default access control enforcement mode.
Value	(string) • **enforcing:** Full NACM enforcement • **permissive:** All but read access control checking • **disabled:** No access control checking, except for NACM secure/very-secure tagged objects • **off:** No access control checking
CLI Override	--**access-control** parameter
Default	enforcing

agt_acm_model

Description	The **agt_yuma_system_notifs** field specifies if the **yuma-system** module should be used for system events.
Value	(agt_acm_model_t) • **AGT_ACM_MODEL_IETF_NACM**: Use **ietf-nacm** module for access control model • **AGT_ACM_MODEL_YUMA_NACM**: Use **yuma-nacm** module for access control model
CLI Override	None
Default	AGT_ACM_MODEL_IETF_NACM

agt_superuser

Description	The **agt_superuser** field specifies the name of the superuser account that will be allowed to bypass access control enforcement.
Value	(string)
CLI Override	--**superuser** parameter
Default	None

agt_crypt_hash_prefix

Description	The **agt_crypt_hash_prefix** field specifies the hash algorithm <id> prefix to use in crypt-hash processing.
Value	(string) 1, 5, and 6 are supported. See crypt-hash typedef for details.
CLI Override	None
Default	6

agt_min_password_len

Description	The **agt_min_password_len** field specifies the minimum allowed password length to accept in crypt-hash processing.
Value	uint8 [1 .. 255]
CLI Override	None
Default	8

6.2.5 Protocol Capability and YANG Module Fields

agt_useurl

Description	The **agt_useurl** field specifies that the :url capability should be supported
Value	(boolean)
CLI Override	**--with-url** parameter
Default	true

agt_usevalidate

Description	The **agt_usevalidate** field specifies that the :validate capabilities (1.0 and 1.1) should be supported
Value	(boolean)
CLI Override	**--with-validate** parameter
Default	true

agt_use_ccommit

Description	The **agt_use_ccommit** field specifies that the :confirmed-commit capabilities (1.0 and 1.1) should be supported
Value	(boolean)
CLI Override	None
Default	true

agt_use_cli

Description	The **agt_use_cli** field specifies that the yp-shell CLI protocol be enabled. The server must be compiled and installed using the WITH_CLI=1 make parameter
Value	(boolean)
CLI Override	None
Default	true if compiled WITH_CLI=1, false if not

agt_use_local_transport

Description	The **agt_use_local_transport** field specifies that direct TCP connections will be accepted
Value	(boolean)
CLI Override	None
Default	true if compiled DEBUG=1, false if not

agt_use_netconf

Description	The **agt_use_netconf** field specifies that the NETCONFprotocol be enabled.
Value	(boolean)
CLI Override	None
Default	true

agt_use_yangapi

Description	The **agt_use_yangapi** field specifies that the YANG-API protocol be enabled. The server must be compiled and installed using the WITH_YANGAPI=1 make parameter
Value	(boolean)
CLI Override	None
Default	true if compiled WITH_YANGAPI=1, false if not

agt_use_yuma_arp

Description	The **agt_use_yuma_arp** field specifies that the **yuma-arp** YANG module will be loaded and activated at boot-time
Value	(boolean)
CLI Override	None
Default	true

agt_use_yuma_if

Description	The **agt_use_yuma_if** field specifies that the **yuma-interfaces** YANG module will be loaded and activated at boot-time
Value	(boolean)
CLI Override	None
Default	true

agt_use_yuma_mysession

Description	The **agt_use_yuma_mysession** field specifies that the **yuma-mysession** YANG module will be loaded and activated at boot-time
Value	(boolean)
CLI Override	None
Default	true

agt_use_yuma_proc

Description	The **agt_use_yuma_proc** field specifies that the **yuma-proc** YANG module will be loaded and activated at boot-time
Value	(boolean)
CLI Override	None
Default	true

agt_use_yumaworks_event_filter

Description	The **agt_use_yumaworks_event_filter** field specifies that the **yumaworks-event-filter** YANG module will be loaded and activated at boot-time
Value	(boolean)
CLI Override	None
Default	true

agt_yumaworks_system

Description	The **agt_yumaworks_system** field specifies that the **yumaworks-system** YANG module will be loaded and activated at boot-time
Value	(boolean)
CLI Override	None
Default	true

agt_withdef_enabled

Description	The **agt_withdef_enabled** field specifies the **with-defaults** retrieval modes that are enabled in the server. The retrieval mode for the agt_defaultStyle will be added to this bitmask. A value of zero will remove the 'also-supported' field from the with-defaults capability URI so only the basic-mode will be supported. By default this is 'explicit' basic mode. /* bitmask of the with-defaults enumerations that should be * enabled in the server * explicit: bit0 * trim: bit1 * report-all: bit2 * report-all-tagged: bit3 */ uint8 agt_withdef_enabled;
Value	(uint8)
CLI Override	None
Default	all bits enabled (15)

6.2.6 General Fields

agt_alt_names

Description	The **agt_alt_names** field specifies whether the server will check for alternate names in path expressions in URLs. Affects **YANG-API** protocol. Alternate names are set with the YANG **alt-name** extension.
Value	(boolean)
CLI Override	**--alt-names** parameter
Default	true

agt_conffile

Description	The **agt_conffile** field specifies the name of the default configuration file for CLI parameters.
Value	(string)
CLI Override	**--config** parameter
Default	/etc/yumapro/netconfd-pro.conf

agt_hello_timeout

Description	The **agt_hello_timeout** field specifies how long the server should wait for a <hello> message, after a valid <ncx-connect> message has been received. If the timeout is reached then the session is dropped.
Value	(uint32) • 0 = wait forever • 10 - 3600 = wait this number of seconds
CLI Override	**--hello-timeout** parameter
Default	600

agt_idle_timeout

Description	The **agt_idle_timeout** field specifies how long the server should wait for a <rpc> message, since the <hello> or last <rpc> message has been received. A session will not be checked for idle timeout if a notification subscription is active. If the timeout is reached then the session is dropped.
Value	(uint32) • 0 = wait forever • 10 - 360000 = wait this number of seconds
CLI Override	**--idle-timeout** parameter
Default	3600

agt_indent

Description	The **agt_indent** field specifies the number of spaces to indent for logging
Value	(int32) • 0 - 9 = number of spaces to indent
CLI Override	**--indent** parameter
Default	1

agt_lax_namespaces

Description	The **agt_lax_namespaces** field specifies if strict XML Namespace rule checking will be done or not. If 'true', lax checking will be done. If 'false' then strict checking will be done.
Value	(boolean)
CLI Override	None
Default	true

agt_linesize

Description	The **agt_linesize** field specifies the desired line length when generating logging and message output.
Value	(uint32)
CLI Override	None
Default	72

agt_match_names

Description	The **agt_match_names** field specifies whether how to match names in path expressions in URLs and nmaes in CLI commands. Affects **CLI** and **YANG-API** protocols.
Value	(ncx_name_match_t) • **NCX_MATCH_EXACT**: exact full command match (case, length) • **NCX_MATCH_EXACT_NOCASE**: case-insensitive match (length) • **NCX_MATCH_ONE**: exact partial command only match (case) • **NCX_MATCH_ONE_NOCASE**: case-insensitive partial command only match • **NCX_MATCH_FIRST**: exact partial command first match (case) • **NCX_MATCH_FIRST_NOCASE**: case-insensitive partial command first match
CLI Override	**--match-names** parameter
Default	NCX_MATCH_EXACT

agt_max_sessions

Description	The **agt_max_sessions** field specifies the maximum number of management sessions to allow at once.
Value	(uint32) • 0 = no limit • 1 to 1024 = number of sessions allowed at once
CLI Override	**--max-sessions** parameter
Default	8

agt_sil_getbulk_max

Description	The **agt_sil_getbulk_max** field specifies the maximum number of GETBULK entries that will be requested from a GET2 callback function
Value	(uint32) • 0 = no limit • 1 to max = number of requested getbulk entries
CLI Override	**--max-getbulk** parameter
Default	10

agt_simple_json_names

Description	The **agt_simple_json_names** field specifies whether the server will output name of the module in which the data node is defined
Value	(boolean) • **false**: a namespace-qualified member name will be used for all members of a top-level JSON object and then also whenever the namespaces of the data node and its parent node are different.
CLI Override	**--simple-json-names** parameter
Default	false

agt_message_indent

Description	The **agt_message_indent** field specifies the indentation that the server should use when sending test protocol messages in XML or JSON encoding.
Value	(int32) • -1 == no newlines and no indentation; 1 long line per message • 1 .. 9 == use newlines and increase indentation level by this number of spaces for a child node
CLI Override	**--message-indent** parameter
Default	-1

agt_ports

Description	The **agt_ports** field specifies the TCP port number(s) that the server will permit for NETCONF connections. Note that the /etc/ssh/sshd_config Port parameter must be specified for each port number specified in this array.
Value	(uint8[4]) • up to 4 port numbers can be specified • If any ports are specified then the default will not be used
CLI Override	--**port** parameter
Default	830

agt_restconf_server_url

Description	The **agt_restconf_server_url** field specifies the URL prefix string to use for Location headers. This should specify the protocol (http or https) and the host name of the server. Affects RESTCONF protocol.
Value	(string)
CLI Override	--**restconf-server-url** parameter
Default	http://localhost

agt_restconf_strict_accept

Description	The **agt_restconf_strict_accept** field specifies whether the server will accept only normative Accept headers or not.
Value	(boolean) • **true**: the server will only accept requests with normative Accept header entries specified in the draft-ietf-netconf-restconf-08. The MIME media type must be application/yang, submedia type must match requested resource, and the Accept header must not be empty; otherwise 'not accepteble' error will be returned. • **false**: the server will not validate media types. Only requested output encoding value will be validated.
CLI Override	--**restconf-strict_accept** parameter
Default	false

agt_sil_missing_error

Description	The **agt_sil_missing_error** field specifies whether the server will treat a missing SIL library file as an error or a warning
Value	(boolean) • **true**: treat a missing SIL file as an error • **false**: treat a missingSIL file as a warning
CLI Override	--**sil-missing-error** parameter
Default	false

agt_stream_output

Description	The **agt_stream_output** field specifies whether the server will stream data nodes in protocol messages or not
Value	(boolean) • **true**: Maintain just one output buffer per session and send the buffer to the client when the the message is complete or the buffer is full • **false**: Queue message buffers for each session until the message is complete and then send the entire message at once
CLI Override	None
Default	true

agt_wildcards

Description	The **agt_wildcards** field specifies whether the server will check for the dash '-' character as a wildcard replacement in URI path expressions. This affects the YANG-API protocol.
Value	(boolean)
CLI Override	--**wildcard-keys** parameter
Default	false

agt_with_config_id

Description	The **agt_with_config_id** field indicates if the YumaWorks :config-id capability will be enabled. This is used to help cache device configurations. It is an enterprise capability URI, not a standard YANG module URI.
Value	(boolean)
CLI Override	--**with-config-id** parameter
Default	true

agt_with_warnings

Description	The **agt_with_warnings** field indicates if agt_record_warning will be allowed to set the error-severity field to warning
Value	(boolean)
CLI Override	--**with-warnings** parameter
Default	false

agt_xmlorder

Description	The **agt_xmlorder** field specifies whether the server will enforce strict XML document order for incoming XML messages.
Value	(boolean) • **true**: enforce XML document order • **false**: allow out of order sibling nodes in XML input messages
CLI Override	--**usexmlorder** parameter
Default	false

agt_yangapi_server_url

Description	The **agt_yangapi_server_url** field specifies the URL prefix string to use for Location headers. This should specify the protocol (http or https) and the host name of the server. Affects YANG-API protocol.
Value	(string)
CLI Override	--**yangapi-server-url** parameter
Default	http://localhost

agt_library_mode

Description	The **agt_library_mode** field indicates if the server should operate in YANG module library mode. It will find all the YANG modules and make them available for <get-schema> operations.
Value	(boolean)
CLI Override	--**library-mode** parameter
Default	false

6.2.7 Fields to enable/disable specific protocols

agt_with_netconf

Description	The **agt_with_netconf** field specifies whether the NETCONF protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled.
Value	(boolean)
CLI Override	--**with-netconf** parameter
Default	true

agt_with_restconf

Description	The **agt_with_restconf** field specifies whether the RESTCONF protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled.
Value	(boolean)
CLI Override	--**with-restconf** parameter
Default	true

agt_with_yang_api

Description	The **agt_with_yang_api** field specifies whether the YANG-API protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled.
Value	(boolean)
CLI Override	--**with-yang-api** parameter
Default	true

agt_with_yp_shell

Description	The **agt_with_yp_shell** field specifies whether the YPSHELL protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled.
Value	(boolean)
CLI Override	--**with-yp-shell** parameter
Default	true

agt_with_yp_coap

Description	The **agt_with_yp_coap** field specifies whether the YP-CoAP protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled.
Value	(boolean)
CLI Override	--**with-yp-coap** parameter
Default	false

agt_with_yp_coap_dtls

Description	The **agt_with_yp_coap_dtls** field specifies whether the the YP-CoAP over DTLS protocol protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled. (Not implemented yet)
Value	(boolean)
CLI Override	--**with-yp-coap-dtls** parameter
Default	false

6.2.8 High Availability specific fields

agt_ha_enabled

Description	The **agt_ha_enabled** specifies whether the YP-HA protocol should be enabled, allowing High Availability Datastore Replication mode to be supported. If this parameter is enabled then the ha-server-key and ha-server pool must be configured.
Value	(boolean)
CLI Override	--**ha-enabled** parameter
Default	false

agt_ha_sil_standby

Description	The **agt_ha_sil_stadby** field specifies whether the edit callbacks such as SIL, SIL-SA and HOOK instrumentation will be invoked if the server is operating in HA standby mode
Value	(boolean)
CLI Override	--**ha-sil-standby** parameter
Default	false

agt_ha_port

Description	The **agt_ha_port** field specifies whether the the YP-CoAP over DTLS protocol protocol will be enabled or not. The incoming connection will be dropped if the protocol is disabled. (Not implemented yet)
Value	(uint16)
CLI Override	--**with-yp-coap-dtls** parameter
Default	8088

agt_ha_server_key

Description	The **agt_ha_server_key** field specifies the magic string the standby server must present to the active server during registration. Used to prevent servers from going the wrong HA pool. If not set then the active server will reject the YP-HA connection. This parameter must be set if the ha-enabled parameter is set to 'true'.
Value	(string)
CLI Override	--**ha-server-key** parameter
Default	none

agt_ha_initial_active

Description	The **agt_ha_initial_active** field specifies the server name for the initial YP-HA active server. This is ignored unless ha-enabled=true.
Value	(string)
CLI Override	**--ha-initial-active** parameter
Default	none

6.3 yp_system_init1

This function is used to initialize the server during phase 1 initialization.

This function is called twice during system initialization:

1. **pre_cli = TRUE**: Called before the CLI and conf file parameters have been processed
2. **pre_cli = FALSE**: Called after the CLI and conf file parameters have been processed

```
/* 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
 */
extern status_t yp_system_init1 (boolean pre_cli);
```

This function should register system callback functions and setup the system.

The first time the function is called the pre_cli flag is true. This is before the CLI and config file parameters have been applied. The second time the function is called the pre_cli flag is false. This is after the CLI and config file parameters have been applied.

Example Phase 1 Initialization Function

This example phase 1 initialization function shows setup of external vendor logging and external access control model 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
 */
status_t yp_system_init1 (boolean pre_cli)
{
    if (pre_cli) {
         // example -- register vendor callback to consume logging output.
        log_vendor_init1_fn(pre_cli); /* Optional */
       log_vendor_extern_register_send_fn(log_vendor_send_fn);
    } else {
```

```
        // example -- external NACM callbacks
        // load module for external module
        // with ncxmod_load_module

        // register the external ACM callbacks
        // this will have no affect unless the
        // yp_system_init_profile fn sets the
        // agt_acm_model to AGT_ACM_MODEL_EXTERNAL
        agt_acm_extern_register_callbacks(acm_extern_rpc,
                                          acm_extern_notif,
                                          acm_extern_write,
                                          acm_extern_read);

         // example -- register vendor callback to consume logging output.
        log_vendor_init1_fn(pre_cli); /* Optional */
    }
    return NO_ERR;
}  /* yp_system_init1 */
```

6.4 yp_system_init2

This function is used to initialize the server during phase 2 initialization. This function is invoked after all phase 1 initialization has been successfully completed.

The first time the function is called the pre_load flag is true. This is before the running configuration has been initialized. The second time the function is called the pre_load flag is false. This is after the running configuration has been initialized with configuration from non-volatile storage and YANG defaults.

This function should initialize global callbacks based configuration settings.

```
/* 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
 */
extern status_t yp_system_init2 (boolean pre_load);
```

Example Phase 2 Initialization Function

```
/* 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
 */

status_t yp_system_init2 (boolean pre_load)
{
    log_debug("\nyp_system init2\n");

    if (pre_load) {
        ;
    } else {
        ;
    }
    return NO_ERR;
}
```

6.5 yp_system_cleanup

This function is used to cleanup the system when the server is shutting down or restarting.

```
/* system cleanup callback
 * this callback is invoked once during agt_cleanup
 */
extern void yp_system_cleanup (void);
```

Example System Cleanup Function

```
/* system cleanup callback
 * this callback is invoked once during agt_cleanup
 */
void yp_system_cleanup (void)
{
    log_debug("\nyp_system cleanup\n");
}
```

7 System Callback Functions

The callback functions described in this section are invoked as part of normal server operation.

They can be registered from the **yp-system** library module or a SIL module. Some internal functions are documented as well. These are needed by the system and multiple callbacks are not supported.

Refer to the file **/usr/share/yumapro/src/libsystem/src/example-system.c** for examples of the callbacks in this section.

7.1 Candidate Reload

The **Candidate Reload** function is an internal system callback that is used to clear some cached validation data when the running configuration is loaded into to candidate datastore.

Callback Template

- Type: Internal use only
- Max Callbacks: 1
- File: **ncx/cfg.h**
- Template: **cfg_reload_candidate_cb_t**
  - Inputs: none
  - Outputs: none
  - Returns: none
- Register: **cfg_register_reload_candidate_cb**
- Unregister: none

```
/* Typedef of the callback */
typedef void (*cfg_reload_candidate_cb_t) (void);
```

7.2 Module Load

The **Module Load** function is a user/system callback that is invoked when a YANG module is loaded into the server. This callback is only invoked for main modules, not submodules.

Callback Template

- Type: Internal and User Callback
- Max Callbacks: NCX_MAX_LOAD_CALLBACKS (8: ncx/ncxconst.h)
- File: **ncx/ncx.h**
- Template: **ncx_load_cbfn_t** (ncx/ncxtypes.h)
  - Inputs:
    - **ncx_module_t *mod**:
      The internal module structure for the module that was just loaded
  - Outputs: none
  - Returns: none
- Register: **ncx_set_load_callback**
- Unregister: **ncx_clear_load_callback**

```
/* Typedef of the callback */
typedef void (*ncx_load_cbfn_t) (ncx_module_t *mod);
```

7.3 Module Unload

The **Module Unload** function is a user/system callback that is invoked when a YANG module is unloaded from the server. This callback is only invoked for main modules, not submodules.

Callback Template

- Type: Internal and User Callback
- Max Callbacks: NCX_MAX_LOAD_CALLBACKS (8: ncx/ncxconst.h)
- File: **ncx/ncx.h**
- Template: **ncx_unload_cbfn_t** (ncx/ncxtypes.h)
  - Inputs:
    - **ncx_module_t *mod**:
      The internal module structure for the module that is being unloaded
  - Outputs: none
  - Returns: none
- Register: **ncx_set_unload_callback**
- Unregister: **ncx_clear_unload_callback**

```
/* Typedef of the callback */
typedef void (*ncx_unload_cbfn_t) (ncx_module_t *mod);
```

7.4 Validate Complete

The **Validate Complete** function is the user/system callback that is invoked after the **Validate Phase** has been processed during the <commit> operation.

If **Validate 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 **Validate Complete** callback is object independent and module independent which means you don't have to link it to the specific object as it's 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: There is not such an operation as <commit> if the default target is set tot <running>. Thus, the **Validate 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: 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_t**:

    Status of the callback function execution
- Register: **agt_cb_validate_complete_register**
- Unregister: **agt_cb_validate_complete_unregister**

```
/* 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);
```

7.5 Apply Complete

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's 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>).

Note: There is not such an operation as <commit> if the default target is set tot <running>. Thus, the **Apply 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: 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_t**:

    Status of the callback function execution
- Register: **agt_cb_apply_complete_register**
- Unregister: **agt_cb_apply_complete_unregister**

```
/* 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);
```

7.6 Commit Complete

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**

```
/* Typedef of the callback */
typedef status_t
    (*agt_commit_complete_cb_t)(agt_commit_type_t commit_type);
```

7.7 NV-Load

The **NV-load** function is a user/system callback that is invoked when the running configuration needs to be read from non-volatile storage. A configuration filespec is passed to this callback that contains the configuration that needs to be saved to non-volatile storage.

Callback Template

- Type: User Callback
- Max Callbacks: 1
- File: **agt/agt.h**
- Template: **agt_nvload_fn_t**
  - Inputs:
    - **ncx_display_mode_t_ * encoding:**
      The address of the return encoding used in the config file: (values other than XML are TBD)
      NCX_DISPLAY_MODE_XML: XML encoding is used
    - **const xmlChar ** filespec:**
      The address of the return filespec of the configuration loaded from non-volatile storage
  - Outputs:
    - ***encoding:**
      Set to the encoding used in the config file: (values other than XML are TBD)
      NCX_DISPLAY_MODE_XML: XML encoding is used
    - ***filespec:**
      Set to a malloced string containing the filespec of the configuration loaded from non-volatile storage. This must be freed by the called if non-NULL.
      Set to NULL if an error returned
      Set to NULL if NO_ERR returned to indicate that the factory config is being loaded
  - Returns: **status_t**:
    Status of the callback function execution
- Register: **agt_register_local_nv_handler**
- Unregister: none

```
/* Typedef of the callback */
typedef status_t
      (*agt_nvload_fn_t) (ncx_display_mode_t *encoding,
                          xmlChar **filespec);
```

7.8 NV-Save

The **NV-save** function is a user/system callback that is invoked when the running configuration needs to be saved to non-volatile storage. A configuration filespec is passed to this callback that contains the configuration that needs to be saved to non-volatile storage.

Callback Template

- Type: User Callback
- Max Callbacks: 1
- File: **agt/agt.h**
- Template: **agt_nvload_fn_t**
  - Inputs:
    - **ncx_display_mode_t_encoding:**
      The encoding used in the config file: (values other than XML are TBD)
      NCX_DISPLAY_MODE_XML: XML encoding is used
    - **const xmlChar * filespec:**
      The filespec of the configuration to save to non-volatile storage
  - Outputs: none
  - Returns: **status_t**:
    Status of the callback function execution
- Register: **agt_register_local_nv_handler**
- Unregister: none

```
/* Typedef of the callback */
typedef status_t
      (*agt_nvsave_fn_t) (ncx_display_mode_t encoding,
                          const xmlChar *filespec);
```

7.9 Config Replay

The **Config Replay** function is a user/system callback that is invoked when the replay configuration procedure is started and finished. A configuration replay is triggered by a user timer callback function.

When the timer function detects that the back-end system process needs to be reloaded with the current configuration, it needs to call the **agt_request_replay()** function to cause the server to start the configuration replay procedure.

Callback Template

- Type: User Callback
- Max Callbacks: 1
- File: **agt/agt.h**
- Template: **agt_replay_fn_t**
  - Inputs:
    - **boolean_is_start:**
      TRUE: the configuration replay procedure is starting
      FALSE: the configuration replay procedure is finished
  - Outputs: none
  - Returns: none
- Register: **agt_register_replay_callback**
- Unregister: none

```
/* Typedef of the callback */
typedef void (*agt_replay_fn_t) (boolean is_start);
```

7.10 Set-Hook API

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.

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. 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);
```

7.11 Transaction-Hook API

A **Transaction Hook** is a function that is invoked within the transaction when an object is modified. The **Transaction Hook** code will be invoked when edits are committed to the **<running>** configuration. This callback will be invoked before a EDIT-1 or EDIT2 callback for the same object.

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);
```

7.12 Start Transaction API

The **Start Transaction** function is the user/system callback that is invoked before any changes to the candidate database will be committed.

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 callback */
typedef status_t
    (*agt_cb_trans_start_t)(agt_cfg_transaction_t *txcb);
```

7.13 Complete Transaction API

The **Transaction Complete** function is the user/system callback that is invoked after the transactions has been processed.

Callback Template

- Type: User Callback
- Max Callbacks: No limit (except available heap memory)
- File: **agt_cb.h**
- Template: **agt_cb_trans_complete_t**
  - Inputs:
    - **txcb** == transaction control block to check
  - Outputs: none
  - Returns: none
- Register: **agt_cb_trans_complete_register**
- Unregister: **agt_cb_trans_complete_unregister**

```
/* Typedef of the callback */
typedef void
    (*agt_cb_trans_complete_t)(agt_cfg_transaction_t *txcb)
```

8 agt_profile Structure

The following structure is used to represent the **agt_profile**:

```
typedef struct agt_profile_t_ {
    /*** BEGIN DATA SENT TO SUBSYSTEMS ***/
    ncx_agttarg_t       agt_targ;
    ncx_agtstart_t      agt_start;
    log_debug_t         agt_log_level;
    log_debug_t         agt_syslog_log_level;

    log_debug_t         agt_pthread_log_level;
    boolean             agt_session_sync_mutex;

    boolean             agt_log_acm_reads;
    boolean             agt_log_acm_writes;
    boolean             agt_validate_all;
    boolean             agt_has_startup;
    boolean             agt_usestartup;   /* --no-startup flag */
    boolean             agt_factorystartup;   /* --factory-startup flag */

    /* --startup-error parameter: stop, continue enums only */
    boolean             agt_startup_error;  /* T: stop, F: check fallback */

    /* --running-error parameter: stop, continue enums only */
    boolean             agt_running_error;  /* T: stop, F: check fallback */

    boolean             agt_logappend;
    boolean             agt_xmlorder;

    boolean             agt_list_deleteall_ok;
    boolean             agt_leaflist_deleteall_ok;

    boolean             agt_stream_output;   /* d:true; no CLI support yet */
    boolean             agt_delete_empty_npcontainers;     /* d: false */
    boolean             agt_notif_sequence_id;  /* d: false */
    boolean             agt_yuma_system_notifs;  /* d: false */
    boolean             agt_ietf_system_notifs;  /* d: true */
    boolean             agt_yumaworks_system;  /* d: true */
    boolean             agt_yumaworks_templates;  /* d: true */

    /* d: true (needs WITH_SUPPORT_SAVE=1) */
    boolean             agt_support_save;

    /* d: true for <term-msg> notification */
    boolean             agt_term_msg;

    /* Yuma REST-API urlselect extra parameters */
    boolean             agt_alt_names;
    boolean             agt_wildcards;
    ncx_name_match_t    agt_match_names;
    agt_transaction_model_t agt_transaction_model;

    const xmlChar      *agt_accesscontrol;

    /* config file: default: /etc/yumapro/netconfd-pro.conf */
    const xmlChar      *agt_conffile;

    /* config dir: default: /etc/yumapro/netconfd-pro.d */
```

```
    const xmlChar      *agt_confdir;

    const xmlChar      *agt_logfile;
    const xmlChar      *agt_startup;
    const xmlChar      *agt_startup_factory_file;
    const xmlChar      *agt_defaultStyle;
    const xmlChar      *agt_superuser;
    const xmlChar      *agt_extern_libspec;
    const xmlChar      *agt_backup_dir;
    const xmlChar      *agt_server_id;

    uint32              agt_eventlog_size;
    uint32              agt_maxburst;
    uint32              agt_hello_timeout;
    uint32              agt_idle_timeout;
    uint32              agt_linesize;
    int32               agt_indent;
    int32               agt_message_indent;
    boolean             agt_usevalidate;          /* --with-validate */
    boolean             agt_useurl;                    /* --with-url */
    boolean             agt_use_ccommit;  /* enable confirmed-comiit */
    boolean             agt_use_yangapi;
    boolean             agt_use_restconf;
    boolean             agt_use_cli;
    boolean             agt_use_netconf;
    boolean             agt_use_local_transport;
    boolean             agt_use_notifications;
    boolean             agt_system_sorted;
    boolean             agt_lax_namespaces;
    agt_acm_model_t     agt_acm_model;
    ncx_withdefaults_t  agt_defaultStyleEnum;
    agt_acmode_t        agt_accesscontrol_enum;
    uint16              agt_max_sessions;
    uint16              agt_max_cli_sessions;
    uint16              agt_subsys_timeout;

    /* these port numbers are for the NETCONF-over-SSH protocol */
    uint16              agt_ports[AGT_MAX_PORTS];

    /* the CoAP port number used only if WITH_COAP and agt_use_coap set */
    uint16              agt_coap_port;
    uint16              agt_coap_dtls_port;
    const xmlChar      *agt_coap_address;

    const xmlChar      *agt_yangapi_server_url;
    const xmlChar      *agt_restconf_server_url;

    boolean             agt_use_yuma_proc;
    boolean             agt_use_yuma_arp;
    boolean             agt_use_yuma_if;
    boolean             agt_use_yuma_mysession;
    boolean             agt_use_yumaworks_event_filter;
    boolean             agt_use_yuma_system;
    boolean             agt_use_rollback_on_error;
    boolean             agt_use_ycontrol;
    boolean             agt_sil_skip_load;
    boolean             agt_log_event_drops;

    /* treat missing SIL libraries as an error, not warning */
    boolean             agt_sil_missing_error;

    /* do not invoke the SIL callback for a key leaf */
    boolean             agt_sil_skip_keys;
```

```
/* save running config for a commit, in case the rollback fails */
boolean             agt_use_rollback_failed_backup;

/* allow a TCP socket instead of an AF_LOCAL socket for connections */
boolean             agt_allow_tcp_socket;

/* use a TCP socket instead of an AF_LOCAL socket for connections */
boolean             agt_use_tcp_socket;

/* location of subsys info file */
const xmlChar      *agt_subsys_info_file;

/* flag to save owner strings in the database */
boolean             agt_save_owners;

/* flag to print JSON leaf-list value on 1 line */
boolean             agt_json_leaf_list_1line;

/* flag to skip generation of the startup XML file when
 * a save_config is done by the server.  Used with the
 * external config mode where the external system database
 * is already up to date so the XML file is not used
 *  Default is false.  Set in yp_system_init_profile.
 */
boolean             agt_save_config_system;

/* bitmask of the with-defaults enumerations that should be
 * enabled in the server
 *    explicit: bit0
 *    trim: bit1
 *    report-all: bit2
 *    report-all-tagged: bit3
 */
uint8               agt_withdef_enabled;

/* Specifies the number of seconds the YPWatcher process
 * will sleep before checking if the netconfd-pro process
 * has died.  Ignored if --no-watcher CLI parameter was used
 */
uint32              agt_watcher_interval;

/* If true, then transactions to the candidate datastore
 * will be recorded in the audit log.
 * If false, then transactions to the candidate datastore
 * will not be recorded in the audit log.";
 */
boolean             agt_audit_log_candidate;

/* If true, the server will invoke the VALIDATE phase
 *  for SIL and SIL-SA callbacks when each edit is made
 *  to the candidate datastore.
 */
boolean             agt_sil_validate_candidate;

/* If 'true' the server will only accept requests with
 * normative Accept header entries specified in the draft
 */
boolean             agt_restconf_strict_headers;

/* If 'true' the server will treat edit-config auto-delete because
 * of false when-stmts as an error instead of silent delete
 */
```

```
    boolean             agt_autodelete_pdu_error;

    /* Specifies the maximum number of getbulk entries to
     * request from a GET2 callback. This value will be used
     * in the get2cb 'max_entries' field.
     * The value 0 is used to indicate there is no max and
     * the GET2 callback can return as many getbulk entries
     * as desired.  This is the default for leaf-list GET2 callbacks
     */
    uint32 agt_sil_getbulk_max;

    /* specifies the string that will be pre-pended to
     * the password before calling crypt_r to generate
     * the hash for the crypt-hash leaf passed with $0$cleartext
     */
    const xmlChar *agt_crypt_hash_prefix;

    /* minimum password length when setting crypt-hash variables */
    uint8 agt_min_passwd_len;

    /* If 'true' then the corresponding protocol will be
     * enabled. Otherwise, the protocol will not be enabled.
     * The incoming connection will be
     * droped if the protocol is disabled.
     */
    boolean             agt_with_netconf;
    boolean             agt_with_restconf;
    boolean             agt_with_yang_api;
    boolean             agt_with_yp_shell;
    boolean             agt_with_yp_coap;
    boolean             agt_with_yp_coap_dtls;
    boolean             agt_with_netconf_tls;

    log_debug_t         agt_audit_log_console_level;
    log_debug_t         agt_audit_log_level;

    boolean             agt_ha_enabled;
    boolean             agt_ha_sil_standby;
    uint16              agt_ha_port;
    const xmlChar      *agt_ha_server_key;
    const xmlChar      *agt_ha_initial_active;

    // leaf-list ha-server not stored here; not sent to subsys

    boolean             agt_simple_json_names;
    boolean             agt_create_empty_npcontainers; /* d: true */

    /*** END DATA SENT TO SUBSYSTEMS ***/

    /* TBD: add this data to subsystems in the near future */

    /* this field indicates if agt_record_warning will
     * be allowed to set the error-severity field to warning
     */
    boolean             agt_with_warnings;

    /* this field indicates the server is operating in library mode
     * It will look for YANG modules but only load them into its
     * library.
     */
    boolean             agt_library_mode;

    /* this field indicates if the :config-id capability is enabled
```

```
     * or not. This is an enterprise URI and at least 1 opensource
     * tool complains it is not a valid YANG module URI
     */
    boolean             agt_with_config_id;

    /* this field indicates that the server should not load or save
     * using the normal APIs during transaction management.
     * The 'start' choice will be ignored (e.g., --no-startup))
     * and the server will not attempt to load a startup-cfg.xml
     * file.  Transactions will not be saved to NV-storage
     * at all.  Any external NV-storage callbacks will be ignored.
     *
     * Use this mode if NV-load and NV-storage are handled
     * internally and not via the startup-cfg.xml file.
     */
    boolean             agt_no_nvstore;

    /* this field indicates whether the NETCONF hello message should
     * conform to the standard and leave out YANG 1.1 modules.
     */
    boolean             agt_with_yang11_hello;      /* d: false */

    /* this field indicates that the IETF Callhome feature is enabled
     * if true and WITH_CALLHOME is built into the image, then
     * the server will attempt to connect to the callhome client servers
     * specified in the callhome config (ietf-server module TBD)
     */
    boolean             agt_with_callhome;

    /* this field specifies the number of seconds to wait after
     * a connect attempt to the callhome server has failed.
     */
    uint16              agt_callhome_retry_interval;

    /* this field specifies the number of retry attempts the server
     * should attempt to the callhome server before giving up.
     * The value 0 indicates the server should never give up.
     */
    uint16             agt_callhome_retry_max;

    /* the callhome-server parameter is handled by agt_callhome */

    /* set the sshd exectuable path for callhome
     * default is /usr/sbin/sshd
     * only set by vendor in agt_init1 phase
     */
    const xmlChar     *agt_sshd_path;

    /* set the netconf subsystem for sshd exectuable path for callhome
     * default is /usr/sbin/netconf-subsystem-pro
     * only set by vendor in agt_init1 phase
     */
    const xmlChar     *agt_subsys_path;

    /* set the sshd_config file to use for sshd exectuable for callhome
     * default is $HOME/.yumapro/ch_sshd_config.<ch-server-name>
     * only set by vendor in agt_init1 phase
     */
    const xmlChar     *agt_sshd_config;


    /* this flag enables/disables the special OpenConfig usage of
     * the YANG pattern-statement. If true then modules named
```

```
     * openconfig-* will be checked as POSIX patterns,
     * not YANG XSD patterns
     */
    boolean             agt_with_ocpattern;

    /* this flag enables FHS file locations for server data files */
    boolean             agt_fileloc_fhs;

    /* this flag indicates the --no-audit-log CLI parameter */
    boolean             agt_no_audit_log;

    /* this enum indicates the --restconf-default-encoding CLI parameter */
    ncx_msg_encoding_t  agt_restconf_default_encoding;

    /* this flag indicates the --startup-error parm is set to fallback */
    boolean             agt_startup_fallback;  /* T: restart, F: continue */

    /* this flag indicates the --running-error parm is set to fallback */
    boolean             agt_running_fallback;  /* T: restart, F: continue */


    /* the SNMP agent parameters */

    /* this flag indicates that the snmp agent should be enabled */
    boolean             agt_with_snmp;

    /* identify the SNMP agent native mode master|subagent */
    ncx_snmp_agt_role_t agt_snmp_agent_role;

    /* identify the SNMP subagent priority, what priority
     * will be used for OID callbacks registration
     */
    uint16              agt_snmp_subagent_priority;

    /* the libcurl variables */
    boolean             agt_useurl_tftp;           /* --with-url-tftp */
    boolean             agt_useurl_ftp;            /* --with-url-ftp */

    /* --sil-delete-children-first CLI parameter */
    boolean             agt_sil_delete_children_first;

    /* --trim-whitespace CLI parameter */
    boolean             agt_trim_whitespace;

    /* --netconf-tls-address parameter */
    const xmlChar       *agt_netconf_tls_address;

    /* --netconf-tls-cerificate parameter */
    const xmlChar       *agt_netconf_tls_certificate;

    /* --netconf-tls-key parameter */
    const xmlChar       *agt_netconf_tls_key;

    /* --netconf-tls-port parameter */
    uint16              agt_netconf_tls_port;

    /* --netconf-tls-trust-store parameter */
    const xmlChar       *agt_netconf_tls_trust_store;

    /* --insecure-ok parameter */
    boolean              agt_insecure_ok;

    /* --cert-default-user parameter */
```

```
    const xmlChar        *agt_cert_default_user;

    /* --errmsg-lang parameter */
    const xmlChar       *agt_errmsg_lang;

    /* --startup-prune-ok parameter */
    boolean              agt_startup_prune_ok;

    /* no CLI parameter!! set to TRUE!
     * change in yp-system library or agt_profile.c
     */
    boolean              agt_startup_create_ok;

    /* --with-canonical parameter */
    boolean              agt_with_canonical;

    /* --with-modtags parameter */
    boolean              agt_with_modtags;

    /* --sil-invoke-for-defaults parameter */
    boolean              agt_sil_invoke_for_defaults;

    /* this flag indicates that the gNMI support should be enabled */
    boolean             agt_with_gnmi;

    /* this flag indicates the server is rnning as yp-controller
     * and not netconfd-pro
     */
    boolean             agt_ypserver_mode;

    /* --sil-prio-reverse-for-deletes parameter */
    boolean              agt_sil_prio_reverse_for_deletes;

    /* --audit-log-events to control audit log content */
    uint32                agt_audit_log_events;

    /* --sil-root-check-first to do root check in edit-config
     * before the SIL validate callbacks are invoked
     * the old (only) behavior is 'false'
     */
    boolean             agt_sil_root_check_first;

    /* No CLI parameter!! Set to TRUE in agt_profile.c
     * the load-config will wait if agt_ncx_load_any_waiting()
     * is true and this parameter is also true
     */
    boolean              agt_sil_wait_sa;

    /* allow maintenance mode to be used */
    boolean              agt_with_maint_mode;

    /* CLI parameter --callhome-reconnect */
    boolean              agt_callhome_reconnect;

    /* enable config=false when-stmt checking for GET1 and GET2
     * callback functions; if false then the callback is expected
     * to check the when-stmt itself and return ERR_NCX_NO_INSTANCE
     * if the when-stmts for the node are false; default is true
     */
    boolean              agt_sil_test_get_when;

    /* use the yuma-time-filter module */
    boolean              agt_yuma_time_filter;
```

```
    /* use the yumaworks-getbulk module */
    boolean              agt_yumaworks_getbulk;

    /* use the yumaworks-ids module */
    boolean              agt_yumaworks_ids;

    /* db-lock used only if WITH_YCONTROL=1 and CLI set */
    boolean              agt_use_db_lock;

    /* max-strlen parameter */
    int32                agt_max_strlen;

    /* with-yumaworks-config-change */
    boolean              agt_with_yumaworks_config_change;

    /* tls-crl-mode parameter */
    agt_crl_mode_t       agt_crl_mode;

    /* tls-crl-missing-ok parameter */
    boolean              agt_crl_missing_ok;

    /* with-nmda */
    boolean              agt_with_nmda;

    /* startup-skip-validation */
    boolean              agt_startup_skip_validation;

    /* convert-subtree-filterparameter */
    boolean              agt_cvt_subtree_filter;

    /* import-version-bestmatch parameter */
    boolean              agt_import_version_bestmatch;

    /* with-yang-patch-running parameter */
    boolean              agt_with_yang_patch_running;


    /****** state variables; TBD: move out of profile ******/

    /* root commit descendant test flags */
    obj_testflags_t     agt_rootflags;

    /* server load-config flags */
    boolean             agt_load_done;
    boolean             agt_load_validate_errors;
    boolean             agt_load_rootcheck_errors;
    boolean             agt_load_top_rootcheck_errors;
    boolean             agt_load_apply_errors;
    boolean             agt_load_factory_fallback;
    boolean             agt_load_def_startup_factory;

    /* Q of malloced ncx_save_deviations_t */
    dlq_hdr_t           agt_savedevQ;

    /* Q of malloced agt_commit_test_t */
    dlq_hdr_t           agt_commit_testQ;

    /* cached location of startup transaction ID file */
    xmlChar            *agt_startup_txid_file;

    /* strdup of socket-address CLI parameter */
    xmlChar            *agt_socket_address;
```

```
    /* listen on this TCP port if TCP socket is enabled */
    uint16              agt_socket_port;

    /* malloced string indicating the confdir pathspec in use */
    xmlChar            *agt_conf_dirspec;
    boolean             agt_confdir_skipped;

    /* saved HA role */
    agt_ha_role_t      agt_ha_role;

    /* saved defer load config flag */
    boolean            agt_defer_load;

} agt_profile_t;
```