YumaPro Yangcli
yp-show API Guide

YANG-Based Unified Modular Automation Tools

Yangcli application Instrumentation Library Development

Version 19.10-12

Table of Contents

1 Preface

1.1 Legal Statements

1.2 Additional Resources

This document assumes you have successfully set up the software as described in the printed 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 ypclient-pro Manual

YumaPro Developer Manual

YumaPro yp-system 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

1.4 Introduction

This document contains a quick reference for the API callback interface for the **yangcli-pro client application**. The external functions allow vendor-specific functions to be used.

2 yp-show External Interface

The **yangcli-pro client application** supports an external vendor library that is loaded at boot-time by the yangcli_pro application. This library code is run in the context of the main yangcli-pro process. It is used to hook vendor-specific functions into the yangcli application,.

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

- initialization
- cleanup
- hook in external yangcli model
- hook in external yangcli interface

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

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

2.1 Mandatory YANG Module Definitions

If vendors add foo-bar to the show commands, then there has to be a YANG module.
They have to make this YANG module and then they have to load the module in their init function. The function **ncxmod_load_module**() is called to load the new YANG module.

```
Example template like this:

static ncx_module_t *mymod = NULL;
status_t foo_init (void)
{
    status_t res = ncxmod_load_module(mod_name,
                                mod_revision, NULL, &mymod);
   ...

}
```

For the show command defined in yangcli-pro.yang, vendors can have their own implementation. They have to register their own functions to overwrite the existing show functions in yangcli-pro.

Use **ycli_show_extern_register_callbacks**() to register the cli show functions at init time.

Example:

```
/*****************************************************************
* FUNCTION ycli_show_extern_register_callbacks
* Register the external callbacks for show implementation
* INPUTS:
```

```
*    module   == YANG module used for this function
*    showfn_keyword == key word used for this function.
*    showfn ==  show function callback
* RETURNS:
*   status of the function registration.
*
*********************************************************************/

ycli_show_extern_register_callbacks (
                        (const xmlChar *) "yangcli-pro",
                        (const xmlChar *) "cache",
                        your_show_cache_function);
```

2.2 Mandatory yp-show API Functions

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

- **yp_show_init**: Initialization function
- **yp_show_cleanup**: Cleanup function

The following example shows **example-show.h** contents from the **libshow/src** directory:

```
#ifndef _H_example_show
#define _H_example_show
/*
 * 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_status
#include "status.h"
#endif


/* yangcli show init callback
 * This callback is invoked to add yangcli_show_cmd_register().
 * It is called at yangcli initialization.
 * INPUTS:none
 * RETURNS:
 *  status; error will abort startup
 */
extern status_t yp_show_init (void);


/* yangcli show cleanup callback
 * this callback is invoked once during yangcli_cleanup
 */
extern void yp_show_cleanup (void);
```

2.3 Terminal Customization APIs

The yp-show library can contain callbacks for altering the display output that is rendered to the standard output in interactive mode. This callback can alter the output before it is displayed to the user.

Command output is processed in the following order:

1. Raw command output using the specified display mode
2. yp-show terminal hook to alter output text
3. pipe command filtering
4. --More-- pagination

2.3.1 Register Callbacks

There are 2 callbacks that are registered, using one function from yangcli_term.h:

```
/********************************************************************
* FUNCTION ycli_register_term_callback
*
* Register a custom callback to process to server response
* or command output before it is processed by Pipe and More
*
* INPUTS:
*   test_callback == test command callback
*   exec_callback == exec command callback
*********************************************************************/
extern void
    ycli_register_term_callback (term_test_cbfn_t test_callback,
                                 term_cbfn_t exec_callback);
```

2.3.2 Test Function

The **term_test_cbfn_t** callback is invoked to determine if the command output should be processed by the “exec” callback.

```
/* external term handler test callback
* FUNCTION term_api_test_fn
*
* This API tests whether the term_api_fn is needed for this
* command or not;
*
* INPUTS:
*   session_cb == session control block
*   command_name == name of command being invoked or replied
*   reply_output == TRUE if being called from reply handler
*                   FALSE if being called from local command output
* RETURNS:
*    TRUE if command output API hook needed
*    FALSE if ommand output API hook not needed
*********************************************************************/
typedef boolean
    (*term_test_cbfn_t) (void *session_cb,
                         const char *command_name,
                         boolean reply_output);
```

2.3.3 Exec Function

The **term_cbfn_t** callback is invoked to alter the terminal output

```
/* external term handler callback
* FUNCTION term_api_fn
*
* This API alters the command or server output so
* yangcli_term can process the More and Pipe commands
*
* INPUTS:
*   session_cb == session control block
*   in_filespec == input filespec for API to process
*   out_filespec == output file for API to write output
* RETURNS:
*    status
*********************************************************************/

typedef status_t
    (*term_cbfn_t) (void *session_cb,
                    const char *in_filespec,
                    const char *out_filespec);
```

2.3.4 Example

The following code can be found in the **libshow/src/example-show.cpp** source file.

Registration:

```
extern "C" status_t yp_show_init (void)
{
    status_t res = NO_ERR;

    log_debug("\nyp_show init\n");

    ycli_register_term_callback(term_api_test_fn, term_api_fn);
    return NO_ERR;
}
```

Example test callback:

```
/* example terminal output test API
 */
static boolean
    term_api_test_fn (void *session_cb,
                      const char *command_name,
                      boolean reply_output)
{
    /* current session control block if needed */
    session_cb_t *cb = (session_cb_t *)session_cb;
    (void)cb;

    if (reply_output == FALSE) {
        return FALSE;
    }

    if (!strcmp(command_name, "get") ||
        !strcmp(command_name, "get-config")) {

        return TRUE;
    }

    return FALSE;

}  /* term_api_test_fn */
```

Example exec callback:

```
/* example terminal output API
 * This API alters the command or server output so
 * yangcli_term can process the More and Pipe commands
 */
```

```
static status_t
    term_api_fn (void *session_cb,
                 const char *in_filespec,
                 const char *out_filespec)
{
    /* current session control block if needed */
    session_cb_t *cb = (session_cb_t *)session_cb;
    (void)cb;

    /* get file into a buffer
     * this is just one way to process the input;
     * a line-by-line approach could be used to save memory
     */
    xmlChar *in_buff = NULL;
    status_t res = ncx_file_to_buffer(in_filespec, &in_buff);
    if (in_buff == NULL || res != NO_ERR) {
        m__free(in_buff);
        return res;
    }

    /* change all the '{' and '}' chars to spaces just
     * as an example of altering the output
     */
    xmlChar *p = in_buff;
    while (*p) {
        if (*p == '{' || *p == '}') {
            *p = ' ';
        }
        p++;
    }

    /* write altered buffer out to a file */
    res = ncx_buffer_to_file(out_filespec, in_buff);

    m__free(in_buff);

    return res;

}  /* term_api_fn */
```

3 Yangcli Interface

The **yangcli-pro client application** supports external yangcli functions.

The **external** callback functions must be registered with yangcli-pro.

3.1 External Show Callback Functions

The function **ycli_show_extern_register_callbacks** in **yangcli/yangcli_show_extern.h** is used by the external code to register its own show callback functions. The show functions are used by the yangcli-pro.

```
/*********************************************************************
* FUNCTION ycli_show_extern_register_callbacks
*
* Register the external callbacks for show implementation
*
* INPUTS:
*    module   == YANG module used for this function
*    showfn_keyword == key word used for this function.
*    showfn ==  show function callback
*    cookie == context pointer (may be null)
* RETURNS:
*   status of the function registration.
*
*********************************************************************/
extern status_t ycli_show_extern_register_callbacks(
                                const xmlChar* module,
                                const xmlChar* keyword,
                                ycli_show_extern_fn_t showfn,
                                void *cookie);
```

If the external method is selected in the yangcli-pro initialization then the callback function MUST be provided.

The function must be registered:

- **show fn**: Invoke the a vendor specific function. The **yangcli_show_extern_fn_t** template is used for this callback.

The following code snippet shows the API template definitions from yangcli/yangcli_show_extern.h.

```
/*********************************************************************
*
* Callback yangcli_show_extern_fn_t to handle external show
* functions.
*
* INPUTS:
*   server_name == The current server name.
*   rpc == RPC method for the show commad being processed.
*   line == CLI line input.
*   session_name == The name of the current session.
*   valset == valset filled in with parameters for the specified RPC
*   mode == help_mode_t(none, brief, normal, full)
*   cookie == context pointer passed in register time, (may be null)
*
```

```
* RETURNS:
*   Status: NO_ERR or error.
*********************************************************************/
typedef status_t
    (*yangcli_show_extern_fn_t) (const xmlChar *server_name
                                 obj_temp_t    *rpc,
                                 const xmlChar *line,
                                 const xmlChar *session_name
                                 const val_value_t *valset,
                                 help_mode_t mode,
                                 void *cookie);
```

3.2 Example External Yangcli

The following example code is available in **example-show.c,** found in the **libshow/src** directory. It shows some dummy external 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-show.c

  Example External Yangcli Library

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

#include <xmlstring.h>

#include "procdefs.h"
#include "yangcli.h"
#include "example_show.h"
#include "yangcli_show_extern.h"
#include "yangcli_util.h"
#include "dlq.h"
#include "example-show.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 Hooks ******************/

/*********************************************************************
* FUNCTION show_fn
*
* Call back function invoked by yangcli-pro application.
*
* INPUTS:
*   server_name == server name to use.
*   rpc == RPC method for the show commad.
*   line == CLI line input.
*   session_name == the current session name.
*   valset == valset filled in with parameters for the specified RPC.
```

```
*   mode == help_mode_t(none, brief, normal, full)
*   cookie == context pointer (may be null)
*
* RETURNS: NO_ERROR or error.
*
*********************************************************************/
static status_t show_fn (const xmlChar *server_name,
                         obj_temp_t   *rpc,
                         const xmlChar *line,
                         const xmlChar *session_name,
                         const val_value_t *valset,
                          help_mode_t mode,
                         void* cookie);
{
    log_debug("\n\nshow_fn is called in external library.");

    log_debug("\nshow_extern: return OK\n");

    return NO_ERR;

} /* show_fn */



/***************  Required Yancli Show Library Hooks ******************/

static ncx_module_t *mymod = NULL;

/* yangcli show init callback
 *
 * INPUTS:
 * RETURNS:
 *  status; error will abort startup
 */
status_t yp_show_init (void)
{
    log_debug("\nyp_show init\n");
    status_t res = NO_ERR;

    /* Load example-fan.yang
     * Call ncxmod_load_module(mod_name, mod_revision, NULL, &mymod);
     */
    res = ncxmod_load_module((const xmlChar *)"example-fan",
                             (const xmlChar *)"2014-12-05",
                              NULL, &mymod);

    /*
     * Register a show function with yangcli-pro :
     * module name: example-fan for example-fan.yang
     * key word: fan
     * function: show_fan
     */

    /* Create a cookie context if any or NULL */
    void* cookie = NULL;
    if (res == NO_ERR){
       ycli_show_extern_register_callbacks (
                         (const xmlChar *)"example-fan",
                         (const xmlChar *)"fan",
                         show_fan,
                         cookie);
    } else {
       log_debug("\n\nyp_show_init: return ERROR\n");
```

```
    }

    return res;
}


/* yangcli show cleanup callback
 * this callback is invoked once during yangcli_cleanup
 */
void yp_show_cleanup (void)
{
    log_debug("\nyp_show cleanup\n");
}


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

4 yp-yangcli-show Library

The **yp-yangcli-show** library contains vendor code for global APIs that are not specific to 1 YANG module.

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

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

4.1 Copy and Setup libshow subtree

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

The libshow 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-show-example to yp_show
# in the SUBDIR_NM macro below
#
#  SUBDIR_NM=yp_show
#
SUBDIR_NM=yp_show-example
```

Change the string **yp-show-example** to **yp-show** in the src/Makefile in the copy of libshow.

If vendors add foo-bar to the show commands, then there has to be a YANG module.

They have to make this YANG module and then they have to also load the module in their init function.

The function **ncxmod_load_module**() is called to load the new YANG module.

```
Example template like this:

static ncx_module_t *mymod = NULL;
status_t foo_init (void)
{
    status_t res = ncxmod_load_module(mod_name,
                                mod_revision, NULL, &mymod);
   ...
   ...
}
```

4.2 yp_show_init

This function is used to initialize the yangcli application during yangcli-pro initialization.

```
/* yangcli show init callback
 *
 * INPUTS:
 *
 * RETURNS:
 *  status; error will abort startup
 */
extern status_t yp_show_init (void);
```

This initialization function should load vendor's own YANG module, register the show callback functions, and setup the yangcli show in yp_show_init

4.3 yp_show_cleanup

This function is used to cleanup the yangcli show when the yangcli application is shutting down or restarting.

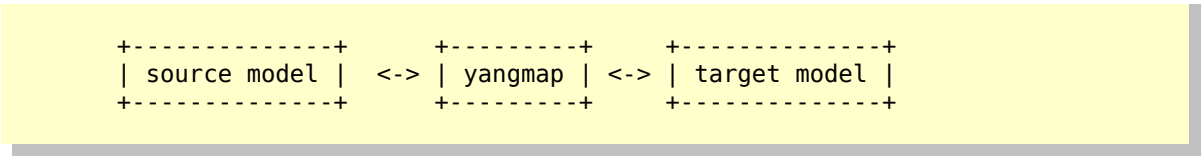
```
/* yangcli show cleanup callback
 * this callback is invoked once during yangcli_cleanup
 */
extern void yp_show_cleanup (void);
```

Example yangcli show Cleanup Function

```
/* yangcli show cleanup callback
 * this callback is invoked once during yangcli_cleanup
 */
void yp_show_cleanup (void)
{
    log_debug("\nyp_show cleanup\n");
}
```

5 yangmap External Interface

The yangmap feature allows one YANG model to be mapped to another YANG model to simplify or stabilize the client-facing command interface, in “config term” mode.



Each yangmap file contains the following information

- source module name(s)
- target module name(s)
- automap flag
- set of nodemaps (source to target)
  - set of keymaps (source to target key)
  - set of childmaps (source to target leaf)

When a yangmap is active, the source model will be used if the target models are supported in a session.

If the server advertises all the target modules in the yangmap, then these modules will be hidden in config term mode.

Instead, the source models will be injected into the session and these modules will be used instead of the target modules.

The “show modules” and “show objects” commands will hide the target modules.

5.1 --yangmap CLI Parameter

The --**yangmap** CLI parameter specifies a YANG mapping file that contains a <yangmap> instance document.

This can be placed in the yangcli-pro.conf config file.

This mapping is used to convert the source model to the target model. There is no support at this time to convert the target model to the source model.

```
     leaf-list yangmap {
       description
         "Specifies a yangmap XML file that should be loaded
          into the program. See yumaworks-yangmap.yang for
          details on using YANG mappings in config term mode.";
       type string;   // filespec
     }
```

Example **yangcli-pro.conf:**

```
yangcli-pro {

yangmap /home/lab/yangmaps/test-ex1-yangmap.xml

}
```

5.2 yumaworks-yangmap YANG Module

```
module yumaworks-yangmap {

    namespace "http://yumaworks.com/ns/yumaworks-yangmap";

    prefix "ymap";

    import ietf-restconf { prefix rc; }
    import yuma-types { prefix yt; }

    organization "YumaWorks, Inc.";

    contact
        "Support <support at yumaworks.com>";

    description
       "YANG model mapping control block parameters.

        +--------------+      +---------+     +--------------+
        | source model |  <-> | yangmap | <-> | target model |
        +--------------+      +---------+     +--------------+

        Purpose:
          This module is used as metadata to convert
          data instances from a source data model to
          a target data model.
            - present simplified model for CLI
            - support a new module with objects that can
              map to existing objects.

          The structure of this metadata allows 1:N model mapping
          of container and list data nodes.

          Child nodes of containers and lists can be mapped to
          multiple target nodes in the target model.

        Terms:
          source model: The data model that is visible to the
          yangcli user in config term mode. This model is for
          client use and does not exist on the server.

          target model: The data model that is implemented on the
          server. This model is not visible to the client in
          config term mode.

          yangmap: a set of mappings to convert a source schema
          node to a target schema node. A source data node can
          be mapped to multiple target nodes.

          key stack: Each data node can be identified by its
          absolute schema-node-identifier string and a set of
          key values called the 'key stack'. The stack is comprised
          of all key leafs from top to bottom and left to right.

          ancestor keys: There are zero or more keys within a
          a key stack that represent keys from ancestor nodes.
          These values are fixed for the purposes of retrieval
```

```
         operations.

         local keys: There are zero or more keys within a
         a key stack that represent keys from the current node.
         This is only possible if the current node is a YANG list
         and there is a key statement defined for that list.
         These values are not fixed for the purposes of retrieval
         operations.

      Usage Restrictions:
         Each yangmap must represent the model mapping for at
         least one subtree.

         Multiple subtrees can be represented in a single yangmap.

         The source and terget subtrees must be completely disjoint.

         Only top-level containers and lists used as nodemap
         source objects are supported at this time.
         An entire source data subtree from top to bottom
         must be mapped.

         Top-level choice-stmts are not supported.
         The target node does not have to be a top-level data node,
         but it must be a container or list node.

         Containers and even lists can be removed from the source
         hierarchy. There can be different containers and lists
         in the target source tree.

         Only configuration data nodes are supported at this time.
         Operational data, RPC, action, notification are not yet
         supported.

         Only containers and lists can be mapped as source or target
         nodes within a nodemap.  Other nodes are allowed within the
         nodemap within a keymap or childmap.

         Only one mapping is supported per node.

         Key mappings must be complete so that any node within
         the source subtree can be mapped to its target node
         without any missing key leaf values.

         If a list is mapped to a container, then only one instance
         of the list will be allowed.

         Leafs should map to other leafs or leaf-list using the
         same data type.  Data type conversions will be attempted
         if not.

         If a leaf-list is mapped to a leaf then only once instace
         is allowed in the leaf-list.

    Copyright (c) 2018 YumaWorks, Inc. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the BSD 3-Clause License
    http://opensource.org/licenses/BSD-3-Clause";

  revision 2018-01-04 {
      description
```

```
      "Initial version";
  }

  // TEMP ADAPTED FROM ietf-sid-file.yang
  typedef schema-node-id {
    type string {
      pattern
        '/[a-zA-Z_][a-zA-Z0-9\-_.]*:[a-zA-Z_][a-zA-Z0-9\-_.]*' +
        '(/[a-zA-Z_][a-zA-Z0-9\-_.]*(:[a-zA-Z_][a-zA-Z0-9\-_.]*)?)*';
    }
    description
      "Identifies a schema-node identifier string for use in the
       YANG mapping. Encoding rules:
       - first node must be in the form /module-name:local-name
       - descendant nodes from the same module must be in the
         form /local-name (no module name present).
       - descendant nodes from a different module (via augments)
         must be in the form /module-name:local-name
       - choice and case statement names must be present";
  }

  typedef child-node-name {
    type string {
      pattern
        '[a-zA-Z_][a-zA-Z0-9\-_.]*(:[a-zA-Z_][a-zA-Z0-9\-_.]*)?';
    }
    description
      "Identifies a child node string for use in the
       YANG mapping. There are two allowed formats:

       module-name:child-name -- used only for a child node
       from a different module namespace than the parent node.

       child-name -- used only for a child node from the
       same module namespace as the parent node.
    ";
  }

  grouping keymap {
    list keymap {
      description
        "There must be one entry for each key leaf in the
         target of the mapping.";
      key key-node;
      leaf key-node {
        type schema-node-id;
        description
          "The leaf data node within the target model
           that corresponds to a key leaf in the keystack.";
      }
      choice keymap-source {
        mandatory true;
        leaf source-node {
          type schema-node-id;
          description
            "The leaf data node within the source model that
             maps to the key-node in this keymap.";
        }
        leaf source-constant {
          type string;
          description
            "The constant value to use for the mapping to
             the key-node.";
```

```
      }
    }
  }
}


grouping nodemap {
  list nodemap {
    key "source-node";
    leaf source-node {
      type schema-node-id;
      description
        "Identifies the source data node for this model map.";
    }
    leaf target-node {
      type schema-node-id;
      mandatory true;
      description
        "Identifies the target data node for this model map.";
    }

    container target-keys {
      description
        "Contains the target model keymaps to be used when
         converting from source to target.";
      uses keymap;
    }

    leaf auto-map {
      type boolean;
      default true;
      description
        "If 'true' then a child node will be mapped
         to a target child node with the same name
         automatically.  If 'false' then explicit childmap
         entries are required for mapped nodes with the same
         name.";
    }
    list childmap {
      description
        "Represents a child node mapping within a parent
         container or list data node. Do not include nested
         complex objects that have their own nodemap entry.";
      key source-child;
      leaf source-child {
        type child-node-name;
        description
          "The name of the child node within the source-node
           that corresponds to this entry. If the node is
           from a different module than the source-node,
           the module name can be present in the name string.

           If this child represents a complex node instead
           of a terminal node, then a separate mapping for
           this node (i.e., source-node represents this node)
           should be present in the nodemap.";
      }
      leaf target-node {
        type schema-node-id;
        description
          "If present, then there is a hard-wired mapping
           from the source data node to a specific target
           data node. This string represents the name of
```

```
               the child node within the target-node.

               If not present, then the source-child node is not
               used within the target-node (i.e., auto-map
               is ignored for the source-child).";
         }
       }
     }
   }

   grouping yangmap {
     container yangmap {
       description
         "Model mapping control block parameters to allow
          different user-facing and instrumentation-facing
          data models.";
       leaf-list source-module {
         type yt:NcxName;
         description
           "Contains the module name for a source model
            module that needs to be loaded to support this
            YANG mapping.";
       }
       leaf-list target-module {
         type yt:NcxName;
         description
           "Contains the module name for a target model
            module that needs to be loaded to support this
            YANG mapping.

            When a session is started, all modules advertised
            by the server will be checked and if all modules
            listed here are present, then this yangmap will
            be enabled for the session.  If the yangmap
            is not valid (e.g., deviations or features from
            server do not match the expected values for this
            yangmap) then an error message will be printed
            and the yangmap will not be used for the session.";
       }
       uses nodemap;
     }
   }

   rc:yang-data yangmap {
     uses yangmap;
   }

}
```

5.3 yangmap Example

This simple example shows how the yangmap feature is configured and used.

5.3.1 Source Model

The YANG module **test-ex2.yang** is used as the source model:

```
module test-ex2 {

    namespace "http://yumaworks.com/ns/test-ex2";

    prefix "e2";

    description
      "Example sourcet module for yangmap
       test-ex1-yangmap.xml creates the foloowing mapping:

       logicalp -> /logical-ports/logical-port
         id     ->                    name
         nest   ->                    nest2
          A     ->                      name
          B     ->                      N2
          C     ->                      N3
      ";

    revision 2018-02-22;

    // test source (client-facing) data model
    list logicalp {
        key "id";   // map to name in test-ex1
        leaf id { type string; }
        list nest {
          key A;
          leaf A { type string; }
          leaf B { type int32; }
          leaf C { type uint32; }
        }
    }

}
```

5.3.2 Target Model

The YANG module **test-ex1.yang** is used as the target model:

```
module test-ex1 {

    namespace "http://yumaworks.com/ns/test-ex1";

    prefix "e1";

    description
      "Example target module for yangmap";

    revision 2018-02-22;

    // test target (hidden from client) data model
    container logical-ports {
      list logical-port {
        key "name";
        leaf name { type string; }

        list nest2 {
          key name;
          leaf name { type string; }
          leaf N2 { type int32; }
          leaf N3 { type uint32; }
        }
      }
    }

}
```

5.3.3 YANGMAP File

The XML file **test-ex1-yangmap.xml** contains the mapping information for this yangmap:

```
<yangmap>
  <source-module>test-ex2</source-module>
  <target-module>test-ex1</target-module>
  <nodemap>
    <source-node>/test-ex2:logicalp</source-node>
    <target-node>/test-ex1:logical-ports/logical-port</target-node>
    <target-keys>
      <keymap>
        <key-node>/test-ex1:logical-ports/logical-port/name</key-node>
        <source-node>/test-ex2:logicalp/id</source-node>
      </keymap>
    </target-keys>
  </nodemap>
  <nodemap>
    <source-node>/test-ex2:logicalp/nest</source-node>
    <target-node>/test-ex1:logical-ports/logical-port/nest2</target-node>
    <target-keys>
      <keymap>
        <key-node>/test-ex1:logical-ports/logical-port/name</key-node>
        <source-node>/test-ex2:logicalp/id</source-node>
      </keymap>
      <keymap>
        <key-node>/test-ex1:logical-ports/logical-port/nest2/name</key-node>
        <source-node>/test-ex2:logicalp/nest/A</source-node>
      </keymap>
    </target-keys>
    <childmap>
      <source-child>B</source-child>
      <target-node>/test-ex1:logical-ports/logical-port/nest2/N2</target-node>
    </childmap>
    <childmap>
      <source-child>C</source-child>
      <target-node>/test-ex1:logical-ports/logical-port/nest2/N3</target-node>
    </childmap>
  </nodemap>
</yangmap>
```

6 External Commands

External commands can be supported by yangcli-pro and yp-shell.

This support is limited to the functionality available to existing yangcli-pro commands:

- Access local data structures
- Access local files
- Generate log output
- Send 1 or 2 commands to the server
- Override or add to the existing reply output processing

6.1 YANG for Command Definitions

The YANG module definition is used for several tasks:

- process command line input, including command name and input parameters
- control tab key completion
- provide help text for commands and input parameters

The YANG syntax is restricted for CLI command purposes:

- must statement validation is not done
- unique statement validation is not done

Example command definition:

```
rpc example-cmd {
  description "Example external command”;
  input {
    leaf parm1 {
      type string;
      description "The first example parameter";
    }
    leaf parm2 {
      type int32;
      description "The second example parameter";
    }
  }
}
```

6.2 API Functions

There are a limited number of API functions available to add an external command.

A yangcli-pro or yp-shell command requires at least 3 components:

1. YANG module “RPC statement” defining the command syntax
2. Callback function to do the command when requested by user input
3. Call to the registration function to add the command into the program

6.2.1 External Command Callback Function

The callback function for the external command must use the “yangcli_command_cbfn_t” definition in yangcli.h:

```
/* Callback template for a local command
 *
 * Handles the command line for the specified command
 *
 * INPUTS:
 *   server_cb == server control block to use
 *   session_cb == session control block to use
 *   rpc == object template for the command
 *   line == input command line to execute
 *   len == offset into the line to start processing
 *          this can be > 0 if the command is the left-hand-side
 *          of an assignment statement
 * RETURNS:
 *    status
 */
typedef status_t
    (*yangcli_command_cbfn_t) (server_cb_t *server_cb,
                               session_cb_t *session_cb,
                               obj_template_t *rpc,
                               const xmlChar *line,
                               uint32 len);
```

The following example callback can be found in example-fan.cpp

```
/********************************************************************
 * FUNCTION do_example_cmd (local RPC)
 *
 * Do Example Command
 *
 * INPUTS:
 *    server_cb == server control block to use
 *    session_cb == session control block to use
 *    rpc == RPC method for the example-cmd command
 *    line == CLI input in progress
 *    len == offset into line buffer to start parsing
```

```
 *
 * RETURNS:
 *   status
 ********************************************************************/
static status_t
    do_example_cmd (server_cb_t *server_cb,
                    session_cb_t *session_cb,
                    obj_template_t *rpc,
                    const xmlChar *line,
                    uint32 len)
{
    (void)rpc;

    status_t res = NO_ERR;
    val_value_t *valset =
        get_valset(server_cb, session_cb, rpc, &line[len], &res);
    if (valset && (res == NO_ERR)) {
        val_value_t *parm1 =
            val_find_child(valset, MODNAME, (const xmlChar *)"parm1");
        if (parm1) {
            log_debug("\nGot parm1=%s", VAL_STR(parm1));
        }

        val_value_t *parm2 =
            val_find_child(valset, MODNAME, (const xmlChar *)"parm2");
        if (parm2) {
            log_debug("\nGot parm2=%d", VAL_INT32(parm2));
        }

        /* do something with the parameters */
    }

    val_free_value(valset);

    return res;

}  /* do_example_cmd */
```

Key steps:

- the input line is parsed. This should be done even if no input parameters to make sure no extra parameters are present.
- a val_value_t tree is produced representing a container of all the input parameters that were parsed
- the parameters are checked with “val_find_child”
- the callback does the work required for the command
- the val_value_t for the input parameters is freed

6.2.2 Register an External Command

The “register_command” function is called from the yp_show_init function to register an external command:

```
/********************************************************************
* FUNCTION register_command
*
* INPUTS:
*    server_cb == server control block to use
*    module == module name containing the RPC method
*    ycli_command_name == RPC method name
*    command_fn == pointer to callback function for this command
*    is_top_cmd == TRUE if opt-level command
*    yangcli_ok == TRUE if OK for yangcli to use it
*    ypshell_ok == TRUE if OK for yp-shell to use it
*    ypserver_ok == TRUE if OK for yp-server to use it
*
* RETURNS:
*    status of the operation
*********************************************************************/
extern status_t
    register_command (
              server_cb_t *server_cb,
              const xmlChar *module,
              const xmlChar *ycli_command_name,
              yangcli_command_cbfn_t command_fn,
              boolean is_top_cmd,
              boolean yangcli_ok,
              boolean ypshell_ok,
              boolean ypserver_ok);
```

Usage Example: (from example-fan.cpp)

```
       server_cb_t *server_cb = get_default_server_cb();
       if (server_cb == NULL){
           res = ERR_NCX_NOT_FOUND;
       } else {
           res = register_command(server_cb,
                                  MODNAME,
                                  (const xmlChar *)"example-cmd",
                                  do_example_cmd,
                                  TRUE,   // is_top_cmd
                                  TRUE,   // yangcli_ok
                                  TRUE,   // ypshell_ok
                                  FALSE); // ypserver_ok (not used)
       }
```

Usage Notes:

- The YANG module identified by “module” must already be loaded
- The command name cannot be a duplicate of any command in yangcli-pro.yang

- The command name cannot be a duplicate of any other external command
- The “is_top_cmd” field must be set to TRUE
- The “ypserver_ok field is ignored but should be set to FALSE

7 Example YANG module

7.1 example-fan.yang

```
module example-fan {

    yang-version 1.1;
    namespace "http://example.com/ns/example-fan";
    prefix fan;
    import yangcli-pro { prefix yp; }

    organization  "Example, Inc.";
    contact "Support <support@example.com>.";

    description
       "Example show command extension for yangcli-pro";

    revision 2018-03-17 {
       description
         "Add example-cmd";
    }

    revision 2017-12-04 {
       description
         "Change to YANG 1.1 to allow empty in case";
    }

    revision 2014-12-05 {
       description
         "Initial version";
    }

    /* add a case to the fan show command, e.g., 'show fan 1'
     * must not clash with other case names in the showtype choice
     */
    augment "/yp:show/yp:input/yp:showtype" {
        case fan {
            leaf fan {
                type union {
                  type empty;
                  type uint32 {
                    range "1 .. max";
                  }
                }
                mandatory true;
                description
                  "The fan number to show or the primary fan if
                   no value is given.";
            }
            leaf diagnostics {
                type empty;
                description
                    "If present, display extra diagnostics info";
            }
        }
        case version {
          leaf version {
            description "Example.com version";
```

```
            type empty;
          }
        }
    }

    rpc example-cmd {
      description
        "Example external command.
         Input parameters are allowed.
         Output parameters are not allowed.
         YANG constraints are not supported.";
      input {
        leaf parm1 { type string; }
        leaf parm2 { type int32; }
      }
    }
}
```