YumaPro gNMI Manual

YANG-Based Unified Modular Automation Tools

YumaPro Google Network Management Interface [gNMI]

Version 18.10-8

Table of Contents

1 Preface

1.1 Legal Statements

1.2 Additional Resources

Other documentation includes:

YumaPro Quickstart Guide

YumaPro Installation Guide

YumaPro User Manual

YumaPro netconfd-pro Manual

YumaPro yangcli-pro Manual

YumaPro ypclient-pro Manual

YumaPro yangdiff-pro Manual

YumaPro yangdump-pro Manual

YumaPro Developer Manual

YumaPro API Quickstart Guide

YumaPro yp-system API Guide

YumaPro yp-show API Guide

YumaPro Yocto Linux Quickstart Guide

YumaPro yp-snmp Manual

To obtain additional support you may contact YumaWorks technical support department:

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 element foo
foo	**netconfd-pro** command or parameter
$FOO	Environment variable FOO
`some text`	Example command or PDU
some text	Plain text
 Informational text	Useful or expanded information
 Warning text	Warning information indicating possibly unexpected side-effects

2 YumaPro gNMI User Guide

YumaPro SDK Architectural Components

2.1 Introduction

This document describes the gNMI (gRPC Network Management Interface) integration within the **netconfd-pro** server and **ypgnmi-app** application. The model-driven configuration and retrieval of operational data using the gNMI CAPABILITIES, GET and SET gRPCs. gNMI version 0.4.0 is supported.

Throughout this specification the following terminology is used:

- **Telemetry** - refers to streaming data relating to underlying characteristics of the device - either operational state or configuration
- **Configuration** - elements within the data schema which are read/write and can be manipulated by the client
- **Target** - the device within the protocol which acts as the owner of the data that is being manipulated or reported on. Typically this will be a **netconfd-pro** server that communicate with the client with help of **ypgnmi-app** application

- **Client** - the device or system using the protocol and the service described in this document to query/modify data on the target, or act as a collector for streamed data. Typically this will be a network management system. The client will contact the **ypgnmi-app** application to query/modify data on the target

2.1.1 Features

The main gNMI service functionality contains the following requests (gRPC):

- **Capabilities** - used by the client and target as an initial handshake to exchange capability information
- **Get** - used to retrieve snapshots of the data on the target by the client
- **Set** - used by the client to modify the state of the target
- **Subscribe** - used to control subscriptions to data on the target by the client

Future revisions of the gNMI specification MAY result in additional services being introduced, and hence an implementation MUST NOT make assumptions that limit to a single service definition.

The **ypgnmi-app** application is capable of handling these requests and transferring them to the **netconfd-pro** server for further processing as well as handle replies from the **netconfd-pro** server and transmit them back to the gNMI clients. The following sections will describe in more details how these requests are handled internally in the **netconfd-pro** and how **ypgnmi-app** application transform original gNMI requests and how they are transmitted to and from the **netconfd-pro**.

2.1.2 Configuration Parameter List

The following configuration parameters are used by **ypgnmi-pro**. Refer to the CLI Reference for more details.

Ypgnmi-app CLI Parameters

parameter	description
--bind-address	Specifies the gNMI server binding
--ca	Specifies the gNMI server CA certificate file
--cert	Specifies the gNMI server certificate file
--fileloc-fhs	Specifies whether the **ypgnmi-app** should use Filesystem Hierarchy Standard (FHS) directory locations to create, store and use data and files
--key	Specifies the gNMI server private key file
--log	Specifies the file path of the application log
--log-console	Directs that log output will be be sent to STDOUT, after being sent to the log file and/or local syslog daemon
--log-level	Controls the verbosity level of messages printed to the log file or STDOUT, if no log file is specified
--server	Specifies the **netconfd-pro** server IP address
--server-id	Specifies the server identifier to use when registering with the **netconfd-pro** server.
--service-id	Specifies the service identifier to use when registering with the **netconfd-pro** server
--subsys-id	Specifies the subsystem identifier to use when registering with the **netconfd-pro** server

2.2 Restrictions for gNMI Protocol

The Phase I of the gNMI protocol integration has several gNMI protocol limitations and does not fully implement the whole set of gNMI protocol features.

2.2.1 Node Values Restrictions. Structured Data Types

The value of a data node (or subtree) is encoded in a **TypedValue** message as a one of field to allow selection of the data type by setting exactly one of the member fields. The possible data types include:

- Scalar types
- Additional types used in some schema languages
- Structured data types (e.g., to encode objects or subtrees)

When structured data is sent by the client or the target in an Update message, it MUST be serialized according to one of the supported by gNMI protocol encodings listed in the Encoding enumeration. The table below lists the gNMI protocol supported encodings and their corresponding **TypedValue** fields. The Phase I of the gNMI protocol integration implements only the Structured data types and the JSON IETF encoding. Other encodings types may be implemented in the feature.

It should be noted that the target never utilizes the Encoding enumeration to declare to the client the type of encoding utilized, hence the client must infer the encoding from the populated **TypedValue** field.

Name	Description	TypedValue field	Encoding Value
JSON	A JSON encoded string	json_val	0
Bytes	An arbitrary sequence of bytes	bytes_val	1
Proto	A Protobuf encoded message using protobuf.any	any_val	2
ASCII	An ASCII encoded string representing text formatted according to a target-defined convention	ascii_val	3
JSON_IETF	A JSON encoded string using JSON encoding compatible with [RFC 7951]	json_ietf_val	4

The Phase I of the gNMI protocol integration implements JSON_IETF encoding. The client MUST support the JSON encoding as a minimum to successfully utilize the target’s replies.

E.g. using the following example data tree:

```
root +
     |
     +-- a +
           |
           +-- b[name=b1] +
                          |
                          +-- c +
                                |
                                +-- d (string)
                                +-- e (uint32)
```

The following serializations would be used:

For /a/b[name=b1]/c:

```
update: <
  path: <
    elem: <
      name: "a"
    >
    elem: <
      name: "b"
      key: <
        key: "name"
        value: "b1"
      >
    >
    elem: <
      name: "c"
    >
  >
  val: <
    json_ietf_val: `{ "d": "AStringValue", "e": 10042 }`
  >
>
```

For /a:

```
update: <
  path: <
    elem: <
      name: "a'
    >
  >
  val: <
    json_ietf_val: `{ "b": [
                    {
                     "name": "b1",
                     "c": {
                     "d": "AStringValue",
                       "e": 10042
                     }
                    }
                  ]
           }`
  >
>
```

2.2.2 GetRequest Message Wildcards Restriction

Path - a set of paths for which the client is requesting a data snapshot from the target. The path specified MAY utilize wildcards. The Phase I of the gNMI protocol integration supports only fully-specified paths. The wildcards in GetRequests may be supported in the feature.

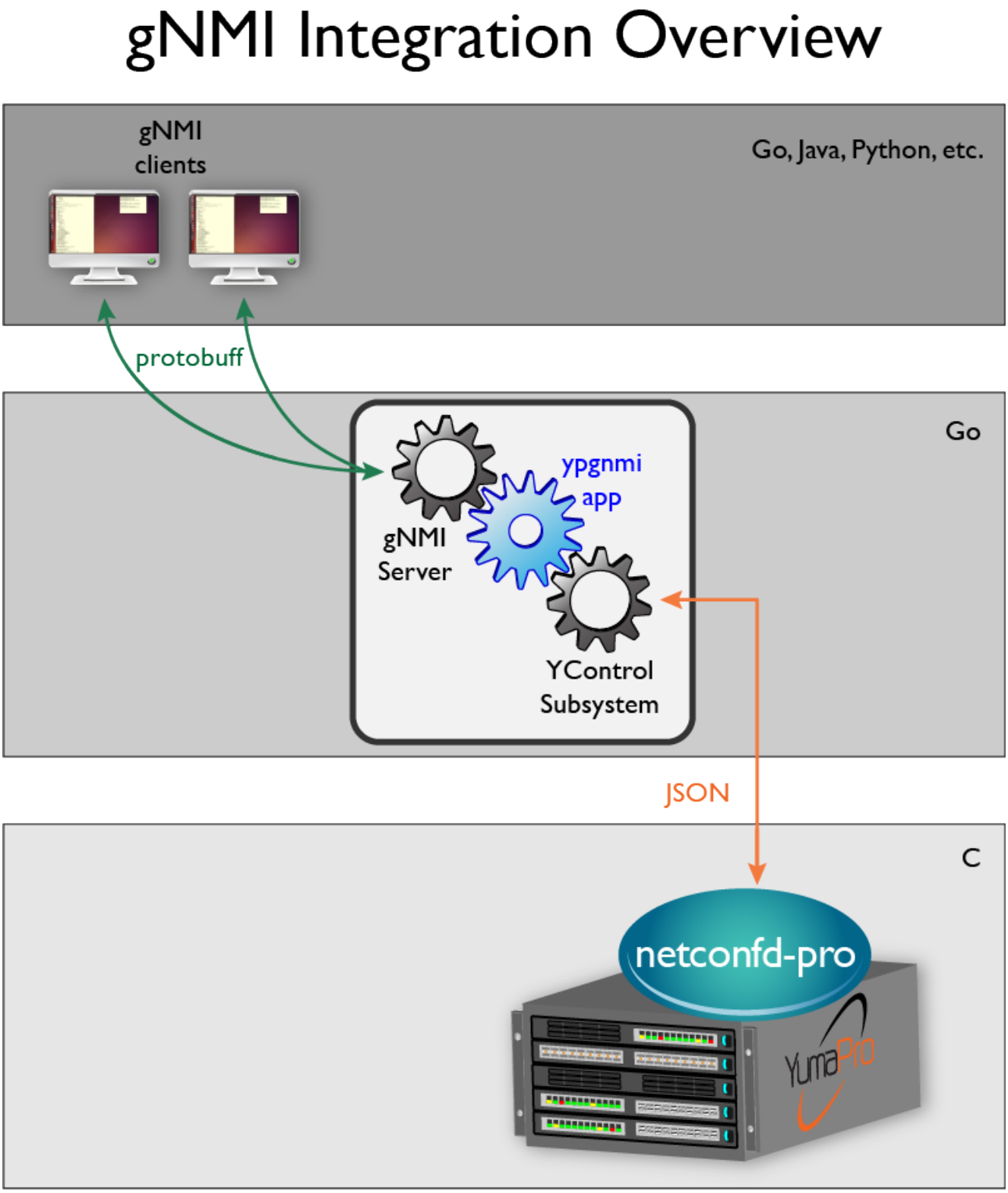
2.2.3 Subscribe Restriction

When a client wishes to receive updates relating to the state of data instances on a target, it creates a subscription via the Subscribe RPC. A subscription consists of one or more paths, with a specified subscription mode. The mode of each subscription determines the triggers for updates for data sent from the target to the client.

All requests for new subscriptions are encapsulated within a SubscribeRequest message - which itself has a mode which describes the longevity of the subscription. A client may create a subscription which has a dedicated stream to return one-off data (**ONCE**); a subscription that utilizes a stream to periodically request a set of data (**POLL**); or a long-lived subscription that streams data according to the triggers specified within the individual subscription's mode (**STREAM**).

The Phase I of the gNMI protocol integration supports only **ONCE** Subscription mode. All other modes may be supported in the feature.

2.3 ypgnmi-app Overview



YumaPro SDK – ypgnmi-app Integration

As illustrated above gNMI clients can be written in any languages that are supported for gNMI clients. The client part is out of the scope of this document and the current gNMI protocol integration does not include client part. The clients communicate to the target (**netconfd-pro**) with help of the **ypgnmi-app** application and send gRPC request to the application.

The **ypgnmi-app** application is written in GO language and talk to the **netconfd-pro** server via socket with XML encoding and acts as a YControl subsystem.

The main role of **yp-gnmi** application is to translate clients requests to XML and contact **netconfd-pro** server and send the replies back to the gNMI clients.

The **ypgnmi-app** application is a YControl subsystem that communicates to the **netconfd-pro** server and also it is a gNMI server that communicates to the gNMI clients. Also, the **ypgnmi-app** application is responsible for encoding conversion between gNMI gRCP to XML that are sent to the **netconfd-pro** server and vise versa.

The messages processing between gNMI client to the **netconfd-pro** server can be split into the following components:

- gNMI clients to the **ypgnmi-app** application processing - includes message parsing and conversion of the messages to the XML message
- The **ypgnmi-app** application to the **netconfd-pro** server messages processing. YControl messages exchange encoded in XML
- The **netconfd-pro** server internal processing – includes subsystem registration, subsystem messages handling and parsing, conversion to RESTCONF RCB for the request processing.
    - YANG-PATCH in case of gNMI SetRequest
    - RESTCONF GET in case of gNMI GetRequest and Subscribe with mode ONCE

The **ypgnmi-app** implements multiple goroutines to manage the replies and the clients. The following goroutines are implemented in the **ypgnmi-app:**

- **Client Manager goroutine**. Responsible for all the gNMI clients connections. After the client contacts the gNMI server, this goroutine register the client, store its information and run all the authentication processing. It verifies the certificates and triggers the connection to be shutdown.
- **Reply Manager goroutine.** This manager is responsible for any already parsed messages from the **netconfd-pro** server or gNMI client, it stores any not processed messages that are ready to be processed. It is responsible to find the corresponding clients that wait for the response and triggers the response procedure.
- **Message Parser goroutine.** It is responsible for all the messages parsing. It parses messages from clients encoded in the protobuf and convert them into XML encoded messages to be send to the **netconfd-pro** server and vice versa.
- **Message Manager goroutine.** This manager is responsible for storing any ready to be processed messages that are going to the **netconfd-pro** server and that are coming back from the server.

All of this managers are goroutines. They run in parallel and asynchronously.

The core of the **ypgnmi-app** is the gNMI server that is build on top of gRPC server. It contains gNMI server code that responsible for the:

- gNMI client to the **netconfd-pro** server communication
- Handle GET/SET/Subscribe/Capability callbacks. Send/get messages from the **netconfd-pro** server and response to the clients.

- Register gNMI server
- Register gRPC server for the protobuf message handling
- Run main Serve loop that handles all the client/server communication

The **ypgnmi-app** application implements the HTTP/2 server over TLS in Go standard package **“golang.org/x/net/http2”**. This package provides authentication, connection, listen, etc handlers that fully fulfill all the requirements for the **ypgnmi-app** application.

The **ypgnmi-app** application has the following startup steps for the gNMI server component:

- Initialize all the prerequisites and parse all the CLI parameters
- Based on the **netconfd-pro** server modules capability create gNMI target
- Open TCP socket to listen for clients requests
- Serve any incoming gNMI messages from gNMI clients and re-transmit them to the **netconfd-pro** server if needed with help of all the goroutine managers.

On the other side the **ypgnmi-app** acts as a YControl subsystem (similar to **db-api-app**), however, it does not terminate after one edit or get request. Instead it continuously listens to the **netconfd-pro** server and keeps the AF_LOCAL or TCP socket open to continue communication whenever it's needed. The communication is terminated only if the **ypgnmi-app** application is terminated or the **netconfd-pro** server terminates.

All the message definitions described in the **yumaworks-yp-gnmi.yang** YANG module and similar to the original gNMI .proto message definitions which makes the conversion easier and faster.

The **ypgnmi-app** application has the following startup steps to initialize connection with the **netconfd-pro** server:

- Initialize all the prerequisites and parse all the CLI parameters
- Open socket and send <ncx-connect> request to the server with
  - transport = netconf-aflocal
  - protocol = yp-gnmi
- Register yp-gnmi service
  - Send <register-request> to the server
  - Register **ypgnmi-app** subsystem and initialize all corresponding code in the **netconfd-pro** server to be ready to handle **ypgnmi-app** application requests
- Create data model structure that will be used for gNMI client/server communication
  - Send <config-request> to the server
  - Feed the gNMI ModelData with all supported modules
- Keep listening socket until terminated

2.3.1 ypgnmi-app Source Files

This section describes the files that are contained in the **yumapro-gnmi** package. The full set of installation sources is installed in **/usr/share/yumapro/src**, if the YumaPro sources are installed.

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

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

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

This application implements a gNMI server that talks to gNMI clients via gRPC calls defined in the gnmi.proto: https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto
and transfer any client requests to the **netconfd-pro** server via NCX socket and YControl protocol using XML/JSON encoded messages.

2.4 Installation

The following sections describe the steps to install and test yp-gnmi.

2.4.1 Prerequisites

To install the Go programming language follow instructions here: https://golang.org/doc/install

Version go1.9 or higher is required. To verify the installation and to verify the version of the installed GO run the following:

```
mydir> go version
go version go1.10 linux/amd64
```

2.4.2 Source Code Installation

If you have installed the YumaPro from source code then you need to build and install using WITH_GNMI=1 WITH_RESTCONF=1 and WITH_YCONTROL=1 build variables.

If you have created your custom workspace directory and would like the **ypgnmi-app** application to be installed in your custom location you will need to set the custom **$GOPATH** and **$GOBIN** Variables. Otherwise, this step is optional and during the installation **$HOME/go** GO workspace will be created and all **ypgnmi-app** application dependencies will be installed there. Follow these steps to setup custom workspace and Variables:

https://github.com/golang/go/wiki/SettingGOPATH

And, you will also need to provide Build Variables GO_PATH=$CUSTOM_GOPATH GO_BIN=$CUSTOM_GOBIN if you created your custom workspace and want the application to be installed there.

```
GO_BIN=<dirspec>: specify the $GOBIN variable dirspec to use when
  building YP-gNMI application. Default is $HOME/go/bin.
  Ignored if PACKAGE_BUILD=1 is also used.

GO_PATH=<dirspec>: specify the $GOPATH variable dirspec to use when
  building YP-gNMI application. Default is $HOME/go.
  Ignored if PACKAGE_BUILD=1 is also used.
```

In this case the **ypgnmi-app** will be installed into your custom **$GOBIN** location. By default the application is installed in the **/usr/bin/.**

2.4.3 Binary Package Installation

If you do not have source code and want to install the YumaPro with a binary package then the application will be installed in the default **/usr/bin** location. If you'd like to use a different directory move the binary to your desired

location. You will have to create your workspace directory, setup **$GOBIN** and **$GOPATH** variables and install all dependencies.

Create your workspace directory, **$HOME/go**. And set the **$GOPATH** environment variable.
https://github.com/golang/go/wiki/SettingGOPATH

```
mydir> mkdir -p ~/go
```

The **$GOPATH** can be any directory on your system. **$HOME/go** is the default **$GOPATH** on Unix-like systems since Go 1.8. Note that **$GOPATH** must not be the same path as your Go installation.

Edit your ~/.bash_profile to add the following lines:

```
export GOPATH=$HOME/go
export GOBIN=$GOPATH/bin
```

Save and exit your editor. Then, source your ~/.bash_profile:

```
mydir> source ~/.bash_profile
```

Using the 'go get' to install the following. Note that in this case **$GOBIN** and **$GOPATH** should be already setup:

```
mydir> go get github.com/aws/aws-sdk-go/aws
mydir> go get github.com/clbanning/mxj
mydir> go get github.com/davecgh/go-spew/spew
mydir> go get github.com/golang/glog
mydir> go get github.com/golang/protobuf/proto
mydir> go get github.com/golang/protobuf/protoc-gen-go/descriptor
mydir> go get -u github.com/kylelemons/godebug/{pretty,diff}
mydir> go get github.com/openconfig/gnmi/proto/gnmi
mydir> go get github.com/openconfig/ygot/experimental/ygotutils
mydir> go get github.com/openconfig/goyang/pkg/yang
mydir> go get -u github.com/openconfig/ygot/{ygot,ytypes}
mydir> go get golang.org/x/net/context
mydir> go get google.golang.org/genproto/googleapis/rpc/code
mydir> go get -u google.golang.org/grpc
mydir> go get -u \
> google.golang.org/grpc/{codes,credentials,metadata,reflection,status}
```

If you get an error similar to:

go/src/github.com/openconfig/ygot/util/schema.go:65: s.Tag.Lookup undefined (type reflect.StructTag has no field or method Lookup)

Upgrade to the latest go version.

2.4.4 Generate the CA Certificates

Generate the client and server certificates. If you have already installed the certificates required for TLS as described in the section “Configure TLS” of the YumaPro SDK Installation Guide make sure copies of the client and server keys and certs are in ~/certs and also the ca.crt is available.

```
     -rw-rw-r--  1 john john  956 Aug  9 10:41 ca.crt
     -rw-rw-r--  1 john john  969 Aug  2 11:00 client.crt
     -rw-rw-r--  1 john john 1704 Aug  2 11:00 client.key
     -rw-rw-r--  1 john john  964 Aug  2 11:01 server.crt
     -rw-rw-r--  1 john john 1704 Aug  2 11:01 server.key
```

Then you can skip to the next section.

If you have not installed TLS certificates follow these steps:

```
     mydir> mkdir ~/certs
     mydir> cp /usr/share/yumapro/util/generate-keys.sh ~/certs
or
     mydir> cp netconfd/util/generate-keys.sh ~/certs
     mydir> cd ~/certs
     certs> ./generate-keys.sh
```

Note: **generate-keys.sh** script contains one line where it configures the server's Common Name and other parameters:

SUBJ="/C=/ST=/L=/O=/CN=your_target_name"

Change it to your target name. By default the value is "restconf". You may keep it for testing, but during the gNMI client requests make sure to specify this target name.

2.4.5 Running ypgnmi-app

Run the server with the setting --with-gnmi=true flag as follows:

```
mydir> netconfd-pro log-level=debug4 --with-gnmi=true
```

Start the **ypgnmi-app** application. Note that you have to provide your certificates to start the application:

```
mydir> man ypgnmi-app
mydir> ypgnmi-app -cert ~/certs/server.crt -ca ~/certs/ca.crt \
       -key ~/certs/server.key
```

After this step the gNMI server starts to listen for any gNMI client requests and will process the requests to the **netconfd-pro** server and will transfer all the replies back.

2.4.6 Running gNMI Client Applications

The gNXI Tools, gRPC Network Management/Operations Interface Tools, provide basic applications to connect to the server and can be used to test the system. Install the gNXI Tools using the 'go get' and 'go install' tools. Refer to https://github.com/google/gnxi:

```
mydir> go get github.com/google/gnxi/gnmi_get
mydir> go install github.com/google/gnxi/gnmi_get
mydir> go get github.com/google/gnxi/gnmi_capabilities
mydir> go install github.com/google/gnxi/gnmi_capabilities
mydir> go get github.com/google/gnxi/gnmi_set
mydir> go install github.com/google/gnxi/gnmi_set
```

To run the **CAPABILITY** client request:

```
mydir> gnmi_capabilities \
       --target_addr 127.0.0.1:10161  \
       --target_name restconf \
       --ca ~/certs/ca.crt \
       --key ~/certs/client.key \
       --cert ~/certs/client.crt
```

To run the **GET** client request:

XPath parameters values are just sample paths of the nodes. Use your real paths that represent your real nodes in YANG modules that you loaded into the netconfd-pro server. Set the --xpath parameter to a real node that you want to retrieve.

```
mydir> gnmi_get \
       --target_addr 127.0.0.1:10161 \
       --target_name restconf \
       --ca ~/certs/ca.crt \
       --key ~/certs/client.key \
       --cert ~/certs/client.crt \
       --xpath "/country[name=usa]" \
       --xpath "person"
```

To run the **SET** client request:

XPath parameters values are just sample paths of the nodes. Use your real paths that represent your real nodes in YANG modules that you loaded into the netconfd-pro server. Set the –delete, --update, --replace parameters paths to real nodes that you want to edit.

```
mydir> gnmi_set \
       --target_addr localhost:10161 \
       --target_name restconf \
       --ca ~/certs/ca.crt \
       --key ~/certs/client.key \
       --cert ~/certs/client.crt \
       --delete /system/openflow/agent/config/max-backoff \
       --delete /system/openflow/agent/config/max-backoff2 \
       --replace /system/clock:@clock-config.json \
       --update /system/clock/config/timezone-name:@clock-config2.json
```

Where clock-config.json Input file to replace container “clock” should look as follows.

The following JSON configurations are just a sample of real nodes configuration. Please use your actual nodes, from YANG modules that you loaded to the netconfd-pro server, here to successfully execute the request.

```
{
  "clock": {
    "timezone-name": "Europe/Stockholm"
  }
}
```

Where clock-config2.json Input file to update a lead “timezone-name” should look as follows:

```
{
    "timezone-name": "ETZ"
}
```

In addition you may use the following gNMI client for sending SET and Subscribe Requests.

Refer to https://github.com/openconfig/gnmi

To run gNMI requests using the following client, for example, **SET** and **Subscribe** request:

```
mydir> > gnmi_cli \
       --address localhost:10161 \
       --server_name restconf \
       --ca_crt ~/certs/ca.crt \
       --client_crt ~/certs/client.crt \
       --client_key ~/certs/client.key \
       --set --delete="/person"
```

```
mydir> > gnmi_cli \
       --address localhost:10161 \
       --server_name restconf \
       --ca_crt ~/certs/ca.crt \
       --client_crt ~/certs/client.crt \
       --client_key ~/certs/client.key \
       --query_type once \
       --query "/uint32.1"
```

The **gnmi_cli** client only sends Subscribe or Set requests. The **ypgnmi-app** application fully supports SET requests. However, only Subscribe requests with ONCE mode are currently supported. The full support of Subscribe requests will be available SOON.

2.4.7 Closing ypgnmi-app

The **ypgnmi-app** can be shut down by typing Ctrl-C in the window that started the application.

If the **netconfd-pro** server is not running when **ypgnmi-app** is started the application will terminate with an error message stating that the **netconfd-pro** server is not running.

If the **netconfd-pro** server is shut down then **ypgnmi-app** will also shutdown.

2.5 gNMI Service definition

2.5.1 gNMI GetRequest

The gNMI GET RPC specifies how to retrieve one or more of the configuration attributes, state attributes, derived state attributes, or all attributes associated with a supported mode from a date tree. A GetRequest is sent from a client to the target to retrieve values from the data tree. A GetResponse is sent in response to a GetRequest.

The following example shows a Get Request on JSON structure:

```
XPath: /oc-if:interfaces/interface[name=Loopback111]
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
encoding: JSON_IETF
++++++++ Recevied get response: ++++++++
notification {
  timestamp: 1521699434792345469
  update {
    path {
      elem {
        name: "oc-if:interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
    }
    val {
      json_ietf_val: "{\n\t\"openconfig-interfaces:name\":\t\"Loopback111\",\n\t\
       "openconfig-interfaces:config\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\"\n\t},\n\t\
       "openconfig-interfaces:state\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\
       "openconfig-interfaces:ifindex\":\t52,\n\t\t\
```

```
      "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\
      "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\
      "openconfig-interfaces:last-change\":\t2018,\n\t\t\
      "openconfig-interfaces:counters\":\t{\n\t\t\t\
      "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\
      "openconfig-interfaces:in-unknown-protos\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\
      "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\
      "openconfig-interfaces:last-clear\":\t2018\n\t\t},\n\t\t\
      "openconfig-platform:hardware-port\":\t\"Loopback111\"\n\t},\n\t\
      "openconfig-interfaces:subinterfaces\":\t{\n\t\t\
      "openconfig-interfaces:index\":\t0,\n\t\t\
      "openconfig-interfaces:config\":\t{\n\t\t\t\
      "openconfig-interfaces:index\":\t0,\n\t\t\t\
      "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\t\
      "openconfig-interfaces:enabled\":\t\"true\"\n\t\t},\n\t\t\
      "openconfig-interfaces:state\":\t{\n\t\t\t\
      "openconfig-interfaces:index\":\t0,\n\t\t\t\
      "openconfig-interfaces:name\":\t\"Loopback111.0\",\n\t\t\t\
      "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\t\
      "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\t\
      "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\t\
      "openconfig-interfaces:last-change\":\t2018,\n\t\t\t\
      "openconfig-interfaces:counters\":\t{\n\t\t\t\t\
      "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\t\
      "openconfig-interfaces:last-clear\":\t2018\n\t\t\t\t}\n\t\t},\n\t\t\
      "openconfig-if-ip:ipv6\":\t{\n\t\t\t\
      "openconfig-if-ip:config\":\t\"false\",\n\t\t\t\
      "openconfig-if-ip:state\":\t\"false\"\n\t\t}\n\t}\n}"
    }
  }
}
```

The following example shows a GetRequest on a leaf on JSON structure:

```
XPath: /oc-if:interfaces/interface[name=Loopback111]/state/oper-status
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "oper-status"
  }
}
encoding: JSON_IETF
++++++++ Recevied get response: ++++++++
notification {
  timestamp: 1521699326012374332
  update {
    path {
      elem {
        name: "oc-if:interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"UP\""
    }
  }
}
```

2.5.2 gNMI SetRequest

The Set RPC specifies how to set one or more configurable attributes associated with a supported model. A SetRequest is sent from a client to a target to update the values in the data tree.

Within an individual transaction (SetRequest) the order of operations is delete, replace, update.

In a SetRequest, only fully-specified (wildcards, and all keys-specified paths are not supported) paths, and only "json_ietf_val" or "json_val" **TypedValue** are supported. JSON keys must contain a YANG-prefix, in which the namespace of the following element differs from parent. The “routed-vlan” element derived from augmentation in openconfig-vlan.yang must be entered as “oc-vlan:routed-vlan”, because it is different from the namespace of the parent node (The parent node prefix is oc-if.).

The total set of deletes, replace, and updates contained in any one SetRequest is treated as a single transaction. If any subordinate element of the transaction fails; the entire transaction will be disallowed and rolled back. A SetResponse is sent back for a SetRequest.

In the case that any operation within the SetRequest message fails, then, the target MUST NOT apply any of the specified changes, and MUST consider the transaction as failed. The target SHOULD set the status code of the SetResponse message to Aborted (10), along with an appropriate error message, and MUST set the message field of theUpdateResult corresponding to the failed operation to an Error message indicating failure. In the case that the processed operation is not the only operation within the SetRequest the target MUST set the message field of the UpdateResultmessages for all other operations, setting the code field to Aborted (10).

The following example shows a SetRequest on JSON structure:

```
Creating UPDATE update for /oc-if:interfaces/interface[name=Loopback111]/config/
XPath: /oc-if:interfaces/interface[name=Loopback111]/config/
++++++++ Sending set request: ++++++++
update {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
  }
  val {
    json_ietf_val: "{\"config\":{\"openconfig-interfaces:enabled\":\"false\"}}"
  }
}
++++++++ Recevied set response: ++++++++
response {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
```

```
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
  }
  op: UPDATE
}
timestamp: 1521699342123890045
```

The following example shows a SetRequest on leaf on JSON structure:

```
Creating UPDATE update for /oc-if:interfaces/interface[name=Loopback111]/config/description
XPath: /oc-if:interfaces/interface[name=Loopback111]/config/description
++++++++ Sending set request: ++++++++
update {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "description"
    }
  }
  val {
    json_ietf_val: "{ \"description\":\"UPDATE DESCRIPTION\" }"
  }
}
++++++++ Recevied set response: ++++++++
response {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
    elem {
```

```
      name: "description"
    }
  }
  op: UPDATE
}
timestamp: 1521699342123890045
```

2.5.3 gNMI JSON_ietf_val

The JSON type indicates that the value is encoded as a JSON string as specified in RFC 7159. Additional types (such as, JSON_IETF) indicate specific additional characteristics of the encoding of the JSON data (particularly where they relate to serialization of YANG-modeled data).

The following is a sample JSON_ietf_val message:

```
val {
  json_ietf_val:"{
   "oc-if:config": {
     "oc-if:description":
       "UPDATE DESCRIPTION"
   }
  }"
}
```

2.5.4 gNMI Error Messages

When errors occur, gNMI server returns descriptive error messages. The following section displays some gNMI server error messages.

The following sample error message is displayed when the path is invalid:

gNMI Error Response:

```
== getRequest:
path: <
  elem: <
    name: "unknown-resource"
  >
>
encoding: JSON_IETF

Get failed: rpc error: code = Code(385) desc = unknown resource
```

3 CLI Reference

The **ypgnmi-app** program uses command line interface (CLI) parameters to control program behavior.

The following sections document all the CLI parameters relevant to this program, in alphabetical order.

3.1 --bind-address

The --**bind-address** parameter specifies the gNMI server binding. This is the address that the gNMI client will use top contact the gNMI server. By default the address is the local host and default port is 10161.

--bind-address parameter

Syntax	**inet:ip-address : inet:port-number**
Default:	:10161
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --bind-address=permissive

3.2 --ca

The --**ca** parameter specifies the gNMI server CA certificate file. The path to the CA certificate should be absolute.

--ca parameter

Syntax	**filespec**
Default:	none
Min Allowed:	1
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --ca=/tmp/ca.crt

3.3 --cert

The **--cert** parameter specifies the gNMI server certificate file. The path to the certificate should be absolute.

--cert parameter

Syntax	**filespec**
Default:	none
Min Allowed:	1
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --cert=/tmp/cert.crt

3.4 --fileloc-fhs

The **--fileloc-fhs** parameter specifies whether the **ypgnmi-app** should use Filesystem Hierarchy Standard (FHS) directory locations to create, store and use data and files. May need to run as root. If false then the server will use $HOME/.yumapro and other file locations to store application data and to access the **netconfd-pro** files.

--fileloc-fhs parameter

Syntax	**boolean**
Default:	false
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --fileloc-fhs=true

3.5 --key

The **--key** parameter specifies the gNMI server private key file. The path to the key should be absolute.

--key parameter

Syntax	**filespec**
Default:	none
Min Allowed:	1
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --key=/tmp/cert.crt

3.6 --log

The **--log** parameter specifies the file path of the application log. Filespec for the log file to use instead of STDOUT. Leave out to use STDOUT for log messages.

--log parameter

Syntax	**filespec**
Default:	none
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --log=/var/log/app-log.log

3.7 --log-console

The --l**og-console** parameter directs that log output will be be sent to STDOUT, after being sent to the log file and/or local syslog daemon. (This assumes that **–-log** parameter is present).

--log-console parameter

Syntax	**boolean**
Default:	none
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --log-console=true

3.8 --log-level

The --**log-level** parameter controls the verbosity level of messages printed to the log file or STDOUT, if no log file is specified.

The log levels are incremental, meaning that each higher level includes everything from the previous level, plus additional messages.

There are 4 settings that can be used:

- **error**: Error messages are printed, indicating problems that require attention.
- **warning**: Warning messages are printed, indicating problems that may require attention.
- **info**: Informational messages are printed, that indicate program status changes.
- **debug**: Debugging messages are printed that indicate program activity.

--log-level parameter

Syntax	**enumeration**: **error** **warning** **info** **debug**
Default:	info
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --log=/var/log/app-log.log \ --log-level=debug

3.9 --server

The --**server** parameter specifies the **netconfd-pro** server IP address. The default is '127.0.0.1' if no value is specified.

--server parameter

Syntax	**inet:ip-address**
Default:	127.0.0.1
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --server=10.10.0.11

3.10 --server-id

The --**server-id** parameter specifies the server identifier to use when registering with the **netconfd-pro** server. The default is 'server1' if no value is specified.

--server-id parameter

Syntax	**string**
Default:	server1
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --server-id=id-12

3.11 --service-id

The --**service-id** parameter specifies the service identifier to use when registering with the **netconfd-pro** server. The default is 'yp-gnmi' if no value is specified.

--service-id parameter

Syntax	**string**
Default:	yp-gnmi
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --service-id=service1

3.12 --subsys-id

The --**subsys-id** parameter specifies the subsystem identifier to use when registering with the **netconfd-pro** server. The default is 'yp-gnmi' if no value is specified.

--subsys-id parameter

Syntax	**string**
Default:	yp-gnmi
Min Allowed:	0
Max Allowed:	1
Supported by:	ypgnmi-app
Example:	ypgnmi-app --subsys-id=subsys1