



Technical Note

How to Access the SmartZone MQTT/GPB API
June 2020

Table of Contents

TABLE OF CONTENTS	2
INTENDED AUDIENCE	3
OVERVIEW	4
What's Covered Here	4
Requirements for this document	4
SmartZone MQTT/GPB API architecture	5
SmartZone Northbound Data Streaming service configuration	6
Installing and configuring the mosquitto broker	7
Decoding the GPB message	10
SmartZone GPB message hierarchy	13
Creating a python script to decode all MQTT/GPB messages	14
Troubleshooting	17
References	19
SUMMARY	19

Intended Audience

This document provides explanation on how to access the Ruckus SmartZone MQTT/GPB API using a mosquitto broker and a python script. The intended audience are system engineers, customers and developers who wish to consume the statistical data provided by SmartZone. The reader should have good knowledge of SmartZone and Python, and a basic understanding of the MQTT/GPB protocols.

For more information on how to configure Ruckus products, please refer to the appropriate Ruckus user guide available in the CommScope support site at <https://www.commscope.com/SupportCenter/>

Overview

This document describes how to access the statistical data provided by SmartZone via the MQTT/GPB API, which can be used to create dashboards, data mining and analytics. That's the same data that is consumed by Ruckus SmartCell Insight (SCI).

The MQTT/GPB API is an interface that allows an external application to receive a stream of statistical data from access points managed by SmartZone. The statistical data includes device information, event records, access point statistics, client statistics, wireless radio/network statistics and rogue AP data. The streaming data is presented in GPB (Google Protocol buffer) format. The external application can use program libraries compiled with the GPB data structure to read the data.

This document is broken into the following sections:

- SmartZone MQTT/GPB API architecture
- SmartZone Northbound Data Streaming service configuration
- Installing and configuring a MQTT broker
- Decoding the GPB message
- Creating a Python script to read the MQTT/GPB messages
- Troubleshooting

What's Covered Here

This document includes an explanation of all building blocks and required protocols, plus detailed descriptions for the steps required to access the SZ MQTT/GPB API.

Requirements for this document

In order to successfully follow the steps in this document, the following equipment and software is required:

- A SmartZone controller running software 5.2 or later, configured with one access point and one WLAN;
- A Windows workstation with Python 3.7.4 and pip installed, Wireshark and Internet access.

SmartZone MQTT/GPB API architecture

SmartZone provides a streaming API which uses the MQTT protocol. The data delivered by this API contains many statistical information about the access point, wireless clients and traffic. That data is transmitted using the GPB (Google Protocol Buffer) protocol. In order to access this API, the best approach is to install a MQTT broker to listen for a connection from SmartZone using TLS-PSK, and create an application to access the broker. By doing that, the more complex authentication and encryption protocols are dealt with by the MQTT broker. Then the application can connect to the MQTT broker without any authentication or encryption.

A popular broker is the **mosquitto** broker. The mosquitto broker for this solution will be configured to accept connections initiated by the SmartZone controller using port 8883 and TLS-PSK. The client application will connect to the broker using port 1883, without authentication or encryption. In this guide we will work with two different clients: the **mosquitto_sub** tool and a python client using the **MQTT Paho python** library. Figure 1 describes the solution architecture to access the SZ MQTT API.

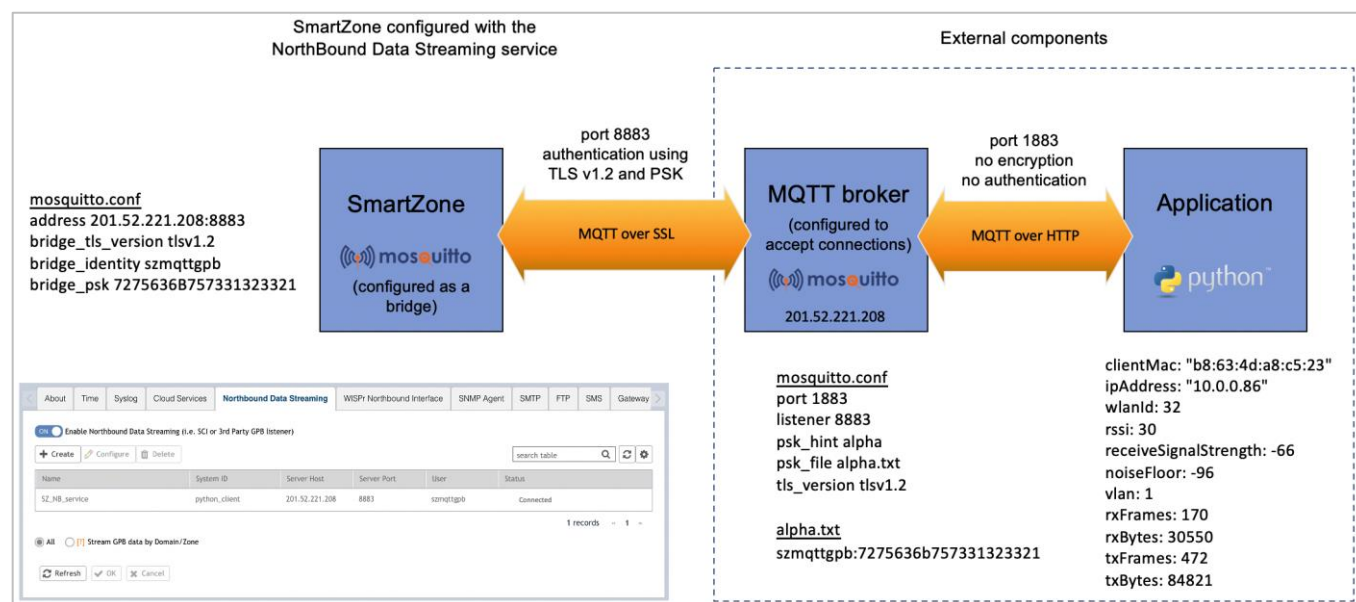


FIGURE 1: SMARTZONE MQTT/GPB API SOLUTION ARCHITECTURE

The solution described in this document runs the external mosquitto broker and the application in the same workstation, but they could be installed in different workstations, for scenarios where more storage is needed, or with different security requirements. Also, the mosquitto broker and the python application could be installed in Windows, Linux or Mac OS workstations.

SmartZone Northbound Data Streaming service configuration

SmartZone needs a Northbound Data Streaming profile to connect to the external MQTT broker. Please follow the next steps to configure the profile:

Log into your SmartZone instance and navigate to System, General Settings, Northbound Data Streaming and click on **Create**.

Fill in the form like the picture below:

Figure 2 shows the 'Edit Northbound Data Streaming Profile' dialog box. The form includes the following fields and options:

- Name: SZ_NB_service
- Server Host: 10.0.0.82
- Server Port: 8883
- User: szmqttgpb
- Password: (masked with dots)
- System ID: python_client
- Data Type: A dropdown menu showing a list of data types: AP (checked), Client (checked), System (checked), and Switch (checked). Under the AP type, several sub-items are listed and checked: ApStatus, ApReport, ApMesh, ApHccddReport, ApRogue, ApAvc, and ApPeer.

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

FIGURE 2: NORTHBOUND DATA STREAMING PROFILE

where:

Name: is a name to identify the profile

Server Host: is the ip address where the external MQTT broker resides

Server Port: is 8883

User: is the identity that will be used to connect to the external MQTT broker

Password: is the password associated with the user identity

System ID: is a name to identify the external application

Data Type: allows the selection of which statistical data will be sent by SmartZone

Click OK to save your profile.

How to Access the SmartZone MQTT/GPB API

Finally, toggle the control **Enable Northbound Data Streaming** to “ON” and click on **OK** to submit the form.

Enable Northbound Data Streaming (i.e. SCI or 3rd Party GPB listener)

☒ ON

+ Create Configure Delete

search table

Name	System ID	Server Host	Server Port	User	Status
SZ_NB_service	python_client	10.0.0.82	8883	szmqttgpb	Disconnected

1 records « 1 »

☒ All ☐ Stream GPB data by Domain/Zone

Refresh OK Cancel

FIGURE 3: ACTIVATING THE PROFILE

The profile will remain “Disconnected” until we install and configure the external MQTT broker in the next section.

Note: SmartZone supports two Northbound Data Streaming profiles, allowing for the statistical data to be sent to two different destinations simultaneously.

Installing and configuring the mosquitto broker

Navigate to <https://mosquitto.org/files/binary/win32/> to download the software and install the mosquitto 1.3.5 broker for Windows. Do not install the software as service. Under the directory where the software was installed, edit the mosquitto.conf configuration file. The configuration file needs to contain the following lines only:

```
port 1883
listener 8883
psk_hint alpha
psk_file alpha.txt
tls_version tlsv1.2
```

Now, under the same directory, create a text file named alpha.txt and add the following line:

```
szmqttgpb:7275636b7573313233
```

where:

szmqttgpb is the **User**: configured in the SmartZone Northbound Data Streaming profile
 7275636b7573313233 is the **Password**: configured in the SmartZone Northbound Data Streaming profile, converted to hexadecimal. In this example, we typed in **ruckus123** for the password in the form at Figure 2. The string **ruckus123** converted to hexadecimal is 7275636b7573313233

Now open a command line, go to the directory where mosquitto was installed and type in the following command to start the broker:

```
mosquitto -c mosquitto.conf -v
```

How to Access the SmartZone MQTT/GPB API

The SmartZone Northbound Data Streaming service should connect to the mosquitto broker. You should see lines like the following:

```

C:\WINDOWS\system32\cmd.exe - mosquitto -c mosquitto.conf -v
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>mosquitto -c mosquitto.conf -v
1590383446: mosquitto version 1.3.5 (build date 08/10/2014 22:54:48.60) starting
1590383446: Config loaded from mosquitto.conf.
1590383446: Opening ipv6 listen socket on port 8883.
1590383446: Opening ipv4 listen socket on port 8883.
1590383447: Opening ipv6 listen socket on port 1883.
1590383447: Opening ipv4 listen socket on port 1883.
1590383454: New connection from 10.0.0.32 on port 8883.
1590383454: New client connected from 10.0.0.32 as MQTT-Client-SZ_NB_service-1590383332584 (c2, k60).
1590383454: Sending CONNACK to MQTT-Client-SZ_NB_service-1590383332584 (0)
1590383454: Received PUBLISH from MQTT-Client-SZ_NB_service-1590383332584 (d0, q2, r0, m1, 'sci-topic')
1590383454: Sending PUBREC to MQTT-Client-SZ_NB_service-1590383332584 (Mid: 1)
1590383454: Received PUBREL from MQTT-Client-SZ_NB_service-1590383332584 (Mid: 1)
1590383454: Sending PUBCOMP to MQTT-Client-SZ_NB_service-1590383332584 (Mid: 1)

```

FIGURE 4: SMARTZONE CONNECTION TO MOSQUITTO BROKER

Return to your SmartZone instance and check the status of your profile. It should show as “Connected” now.

Name	System ID	Server Host	Server Port	User	Status
SZ_NB_service	python_client	10.0.0.82	8883	szmqttgpb	Connected

1 records

FIGURE 5: DATA STREAMING PROFILE IS CONNECTED

Note: If you prefer to use Linux to run the mosquitto broker, please install version 1.4.15 for Ubuntu. Do not install the latest version. At this moment, that is the only version known to work correctly with TLSv1.0, TLS v1.2 and PSK. To install version 1.4.15, please use the following command on a Ubuntu terminal:

```
sudo apt-get install mosquitto:1.4.15-2
```


How to Access the SmartZone MQTT/GPB API

Now let's connect to the broker using the **mosquitto_sub** tool. Open a command line on your Windows workstation, go to the directory where mosquitto was installed and type in the following command:

```
mosquitto_sub -t # -v -d
```

mosquitto_sub is a subscribing client. The parameter **-t #** subscribes to all MQTT topics. Because no host was defined in the command, **mosquitto_sub** assumes it needs to make a connection to the localhost running at port 1883.

You should receive a stream of data similar to the following:

```
C:\WINDOWS\system32\cmd.exe - mosquitto_sub -t # -v -d
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>
C:\Program Files (x86)\mosquitto>mosquitto_sub -t # -v -d
Client mosqsub/18024-LP-MMOLIN sending CONNECT
Client mosqsub/18024-LP-MMOLIN received CONNACK
Client mosqsub/18024-LP-MMOLIN sending SUBSCRIBE (Mid: 1, Topic: #, QoS: 0)
Client mosqsub/18024-LP-MMOLIN received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/18024-LP-MMOLIN received PUBLISH (d0, q0, r0, m0, 'sci-topic', ... (218
sci-topic
python_client:10
10
C0:C5:20:92:A5:2D1 +C0:C5:20:92:A5:2D1QICX7150-C12P*ICX71502
SPR08092b.binx 00000a4-88f0-43de-ba78-01b23f8fa36cb0FFLINer
Client mosqsub/18024-LP-MMOLIN received PUBLISH (d0, q0, r0, m0, 'sci-topic', ... (308
sci-topic
python_client:10
10
$e37452e3-7870-4575-89c1-88bb74c0a7ea1 $Z5200685*||0
$839f87c6-d116-497e-afce-aa8157abd30c1*Super*20
$8b2081d5-9662-40d9-a3db-2a3cf4dde3f71$Administration Domain*1
$51a7cd20-403e-49e9-95b7-628bfba595221 Raspberry"5
Staging Group>e77c7n.ba78-01b23f8fa36c1
```

FIGURE 6: MOSQUITTO_SUB CONNECTION AND MQTT MESSAGES

The bottom part shows a MQTT message. That is not very readable because the MQTT payload is encoded using the GPB protocol. The next section will show the method to decode it.

Decoding the GPB message

While still running the `mosquitto_sub` tool, use Wireshark in your computer to capture one MQTT packet and export its message content to a .bin file. Because we want to capture traffic between two processes in your computer (the mosquitto broker and the mosquitto_sub client), use the Wireshark Npcap loopback adapter. We are interested in capturing one MQTT message as shown below:

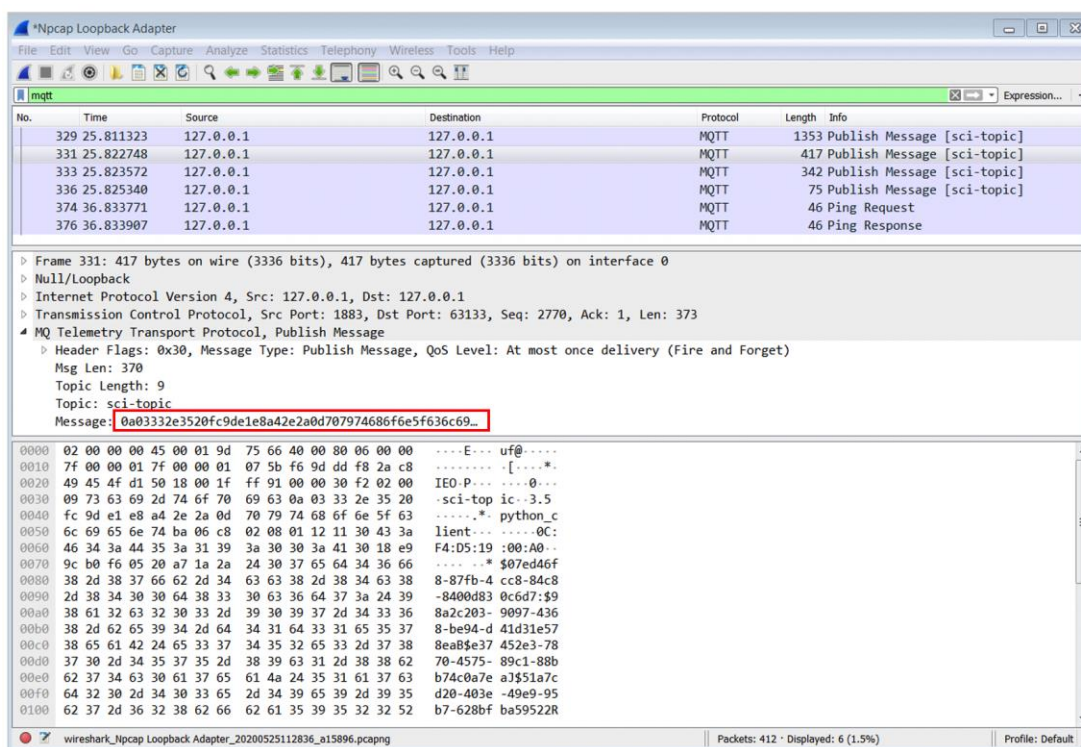


FIGURE 7: MQTT MESSAGE CAPTURED BY WIRESHARK

Do a right-click over the message and click on “Export Packet Bytes”. Save the file in the raw data format. For the following steps we named our file “331.bin”.

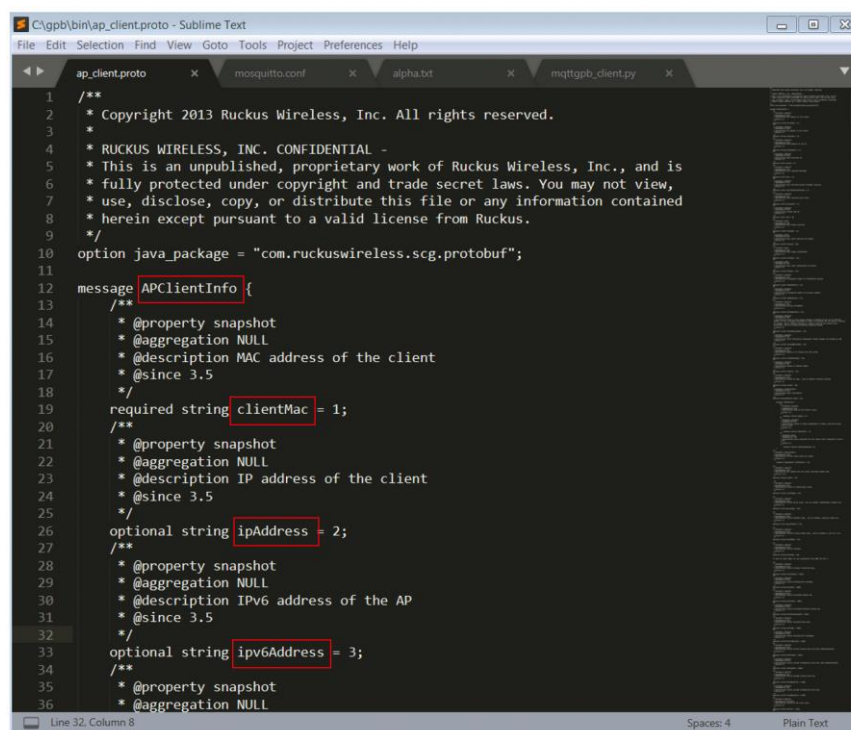
Now, install the command line tools to decode the GPB messages using a .proto file template. Navigate to <https://github.com/protocolbuffers/protobuf/releases/tag/v3.10.0>, then download and install the package `protoc-3.10.0-win64.zip`.

Copy the file “331.bin” to the directory where the file `protoc.exe` was installed.

Download the SmartZone GPB .proto files from the Ruckus support site at <https://support.ruckuswireless.com/software/2392-smartzone-5-2-0-0-699-ga-gpb-proto-google-protobuf-image-for-gpb-mqtt>, unzip the files and copy them over to the same directory where `protoc.exe` was installed. Make sure to copy the google directory too.

How to Access the SmartZone MQTT/GPB API

For the next step, we will only use the **ap_client.proto** file. If you open that file with a text editor, you will see the lines in Figure 8:



```
1  /**
2   * Copyright 2013 Ruckus Wireless, Inc. All rights reserved.
3   *
4   * RUCKUS WIRELESS, INC. CONFIDENTIAL -
5   * This is an unpublished, proprietary work of Ruckus Wireless, Inc., and is
6   * fully protected under copyright and trade secret laws. You may not view,
7   * use, disclose, copy, or distribute this file or any information contained
8   * herein except pursuant to a valid license from Ruckus.
9   */
10 option java_package = "com.ruckuswireless.scg.protobuf";
11
12 message APClientInfo {
13     /**
14      * @property snapshot
15      * @aggregation NULL
16      * @description MAC address of the client
17      * @since 3.5
18      */
19     required string clientMac = 1;
20     /**
21      * @property snapshot
22      * @aggregation NULL
23      * @description IP address of the client
24      * @since 3.5
25      */
26     optional string ipAddress = 2;
27     /**
28      * @property snapshot
29      * @aggregation NULL
30      * @description IPv6 address of the AP
31      * @since 3.5
32      */
33     optional string ipv6Address = 3;
34     /**
35      * @property snapshot
36      * @aggregation NULL
```

FIGURE 8: AP_CLIENT.PROTO FILE

APClientInfo is the message type decoded by the **ap_client.proto** file. **clientMac**, **ipAddress** and **ipv6Address** are a few examples of parameters decoded by this .proto file.

To decode the message inside 331.bin using the .proto file, open a command line in your Windows workstation, move to the directory where the protoc.exe utility was installed, and type in the following command:

```
protoc --decode APClientInfo ap_client.proto < 331.bin
```

How to Access the SmartZone MQTT/GPB API

You should obtain a result like the following:



```
C:\WINDOWS\system32\cmd.exe
C:\gpb\bin>
C:\gpb\bin>
C:\gpb\bin>
C:\gpb\bin>
C:\gpb\bin>
C:\gpb\bin>protoc --decode APClientInfo ap_client.proto < 331.bin
[libprotobuf WARNING T:\src\github\protobuf\src\google\protobuf\compiler\parser
to2":' or 'syntax = "proto3";' to specify a syntax version. (Defaulted to pro
clientMac: "3.5"
wlanId: 1293438716
5: "python_client"
103 {
  1: 1
  2: "0C:F4:D5:19:00:A0"
  3: 1590431337
  4: 3367
  5: "07ed46f8-87fb-4cc8-84c8-8400d830c6d7"
  7: "98a2c203-9097-4368-be94-d41d31e578ea"
  8: "e37452e3-7870-4575-89c1-88bb74c0a7ea"
  9: "51a7cd20-403e-49e9-95b7-628bfba59522"
  10: "839f87c6-d116-497e-afce-aa8157abd30c"
  11 {
    6: 48
    6: 48
    6: 48
    6: 48
    5: 0x30303030
    5: 0x30303030
    5: 0x30303030
    5: 0x30303030
    6: 48
    6: 48
    6: 48
    6: 48
  }
  12: "Super"
  13: "Debian"
  14: "default"
  15: "Raspberry"
  16: 1590431335
  17: 180
  18: "H-510A"
  19: "501602002227"
}
C:\gpb\bin>
```

FIGURE 9: DECODED GPB MESSAGE

SmartZone GPB message hierarchy

The messages sent by SmartZone MQTT/GPB API have two parts: a header, which is defined by the **sci-message.proto** file, followed by four groups of 18 different statistical data messages blocks, each one using one or more different .proto files. The .proto files are organized according to the hierarchy at Figure 10:

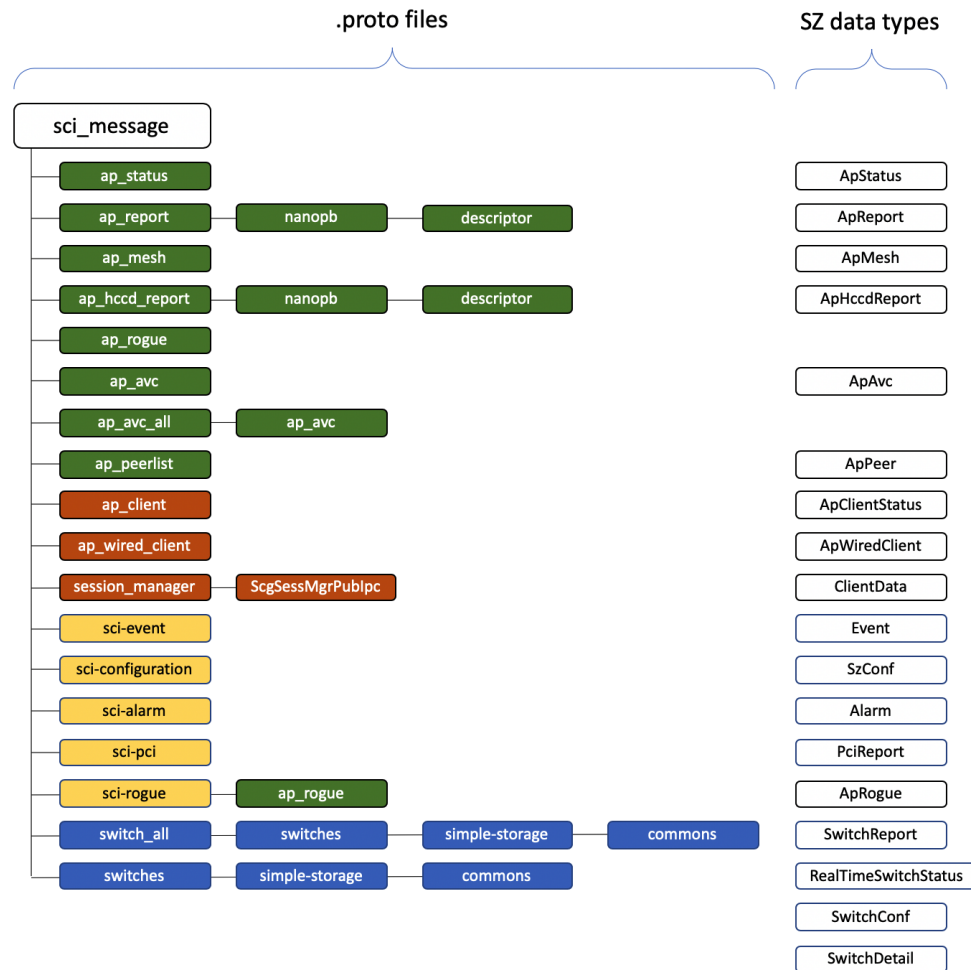


FIGURE 10: .PROTO FILES HIERARCHY AND SZ DATA TYPES

The .proto files descriptions are associated with the data types used by SmartZone. Using the Northbound Data Stream profile, you can define which data types are sent by SmartZone (see Figure 2). For instance, if you select the data type **ApReport**, you need to use the .proto files **ap_report**, **nanopb** and **descriptor** to decode the GPB message.

Creating a python script to decode all MQTT/GPB messages

This document assumes you already have python and pip installed in your computer. To make our life easier, let's install a python library to make mqtt connections. Open a command line and use the following command to install the **paho mqtt** library:

```
pip install paho-mqtt
```

We also need a way to use the .proto files in a python script. The **protoc.exe** utility that we installed in the section “**Decoding the GPB Message**” will be used to compile the required python classes from the .proto files.

We need to compile classes for all .proto files. Go to the directory where all .proto files have been copied, and type in the following commands:

```
protoc -I=. --python_out=. .\ap_avc_all.proto
protoc -I=. --python_out=. .\ap_avc.proto
protoc -I=. --python_out=. .\ap_client.proto
protoc -I=. --python_out=. .\ap_hccd_report.proto
protoc -I=. --python_out=. .\ap_mesh.proto
protoc -I=. --python_out=. .\ap_peerlist.proto
protoc -I=. --python_out=. .\ap_report.proto
protoc -I=. --python_out=. .\ap_rogue.proto
protoc -I=. --python_out=. .\ap_status.proto
protoc -I=. --python_out=. .\ap_wired_client.proto
protoc -I=. --python_out=. .\commons.proto
protoc -I=. --python_out=. .\nanopb.proto
protoc -I=. --python_out=. .\ScgSessMgrPubIpc.proto
protoc -I=. --python_out=. .\sci-alarm.proto
protoc -I=. --python_out=. .\sci-configuration.proto
protoc -I=. --python_out=. .\sci-event.proto
protoc -I=. --python_out=. .\sci-message.proto
protoc -I=. --python_out=. .\sci-pci.proto
protoc -I=. --python_out=. .\sci-rogue.proto
protoc -I=. --python_out=. .\session_manager.proto
protoc -I=. --python_out=. .\simple-storage.proto
protoc -I=. --python_out=. .\switch_all.proto
protoc -I=. --python_out=. .\switches.proto
```

How to Access the SmartZone MQTT/GPB API

Every command will create a different python class ending in **_pb2.py**. You can disregard the messages

"No syntax specified for the proto file: xxxxx.proto. Please use 'syntax = \"proto2\":' or 'syntax = \"proto3\":' to specify a syntax version. (Defaulted to proto2 syntax.)

Copy all the **_pb2.py** files to the directory where your python scripts are.

Now create a python script named **main.py** in the same directory as other scripts, including the following lines:

```
import paho.mqtt.client as mqtt #import the mqtt library
import sci_message_pb2 #import all SZ .proto classes compiled by the protoc tool

#read incoming messages
def on_message(client, userdata, message):
    scim = sci_message_pb2.SciMessage().FromString(message.payload)
    print(scim)
    # print(scim.apClient)
    # print(scim.apClient.ap)
    # print(scim.apClient.clients[0].clientMac)
    # print(scim.apReport)
    # print(scim.apWiredClient)
    # print(scim.apStatus)
    # print(scim.switchDetailMessage)
    # print(scim.switchConfigurationMessage)
    # print(scim.apRogue)

#log function
def on_log(client, userdata, level, buf):
    print("log: ",buf)

broker_address="localhost"

client = mqtt.Client("P1") #create new instance
client.on_message=on_message #attach function to message callback
#client.on_log=on_log

print("connecting to mosquitto broker")
client.connect(broker_address) #connect to mosquitto broker

print("Subscribing to sci-topic")
client.subscribe("sci-topic")

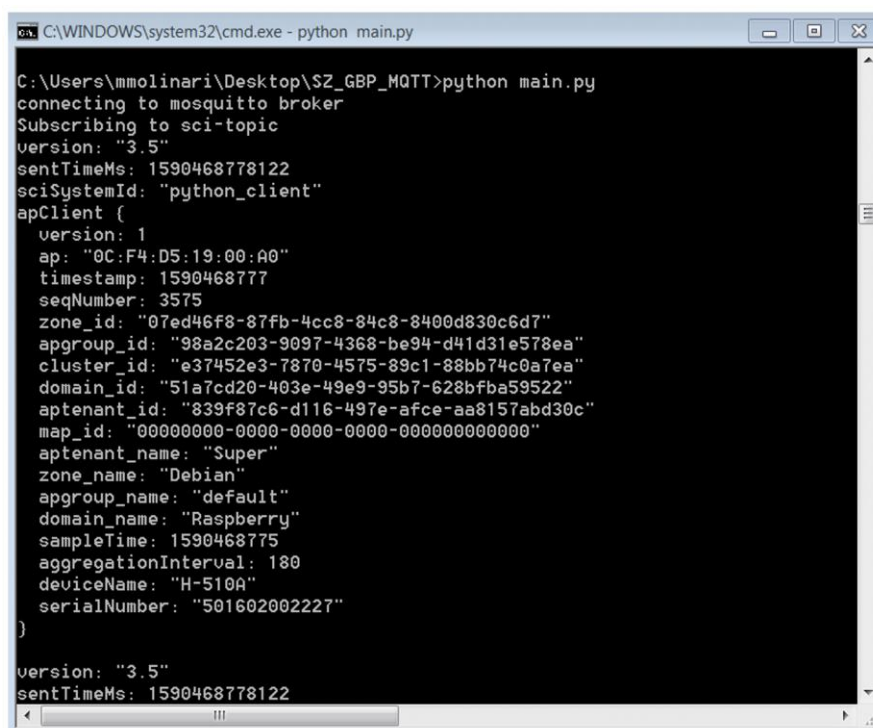
client.loop_forever() #starts a thread on the mqtt client which reads the message buffers
```

The first line imports the paho library to make mqtt connections. The second line imports the main class created by the **protoc.exe** utility. The class **sci_message_pb2** will call the ther **_pb2** classes automatically.

Use the following command to run the script:

```
python main.py
```


It may take a few minutes for the decoded messages might to appear. You will see results similar to Figure 11.



```
C:\WINDOWS\system32\cmd.exe - python main.py
C:\Users\mmolinar\ Desktop\SZ_GBP_MQTT>python main.py
connecting to mosquitto broker
Subscribing to sci-topic
version: "3.5"
sentTimeMs: 1590468778122
sciSystemId: "python_client"
apClient {
  version: 1
  ap: "0C:F4:D5:19:00:A0"
  timestamp: 1590468777
  seqNumber: 3575
  zone_id: "07ed46f8-87fb-4cc8-84c8-8400d830c6d7"
  apgroup_id: "98a2c203-9097-4368-be94-d41d31e578ea"
  cluster_id: "e37452e3-7870-4575-89c1-88bb74c0a7ea"
  domain_id: "51a7cd20-403e-49e9-95b7-628bfba59522"
  aptenant_id: "839f87c6-d116-497e-afce-aa8157abd30c"
  map_id: "00000000-0000-0000-0000-000000000000"
  aptenant_name: "Super"
  zone_name: "Debian"
  apgroup_name: "default"
  domain_name: "Raspberry"
  sampleTime: 1590468775
  aggregationInterval: 180
  deviceName: "H-510A"
  serialNumber: "501602002227"
}
version: "3.5"
sentTimeMs: 1590468778122
```

FIGURE 11: MAIN.PY RESULTS

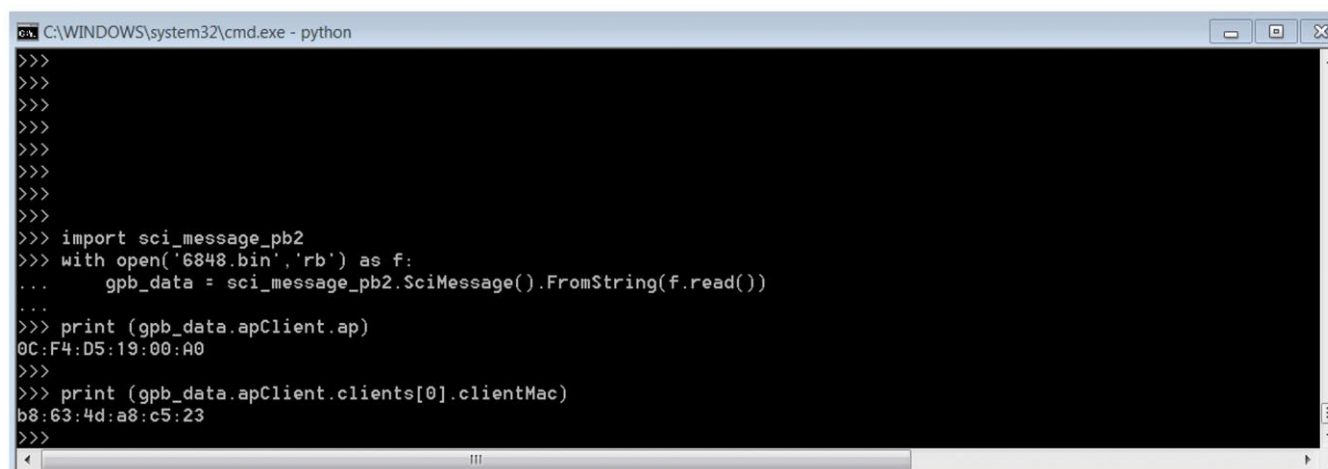
The line `print(scim)` prints all data types sent by SmartZone. The lines after that can be used to selectively print different data types. For instance, use these lines to print only the data types related to switches:

```
def on_message(client, userdata, message):
    scim = sci_message_pb2.SciMessage().FromString(message.payload)
    # print(scim)
    # print(scim.apClient)
    # print(scim.apClient.ap)
    # print(scim.apClient.clients[0].clientMac)
    # print(scim.apReport)
    # print(scim.apWiredClient)
    # print(scim.apStatus)
    print(scim.switchDetailMessage)
    print(scim.switchConfigurationMessage)
    # print(scim.apRogue)
```


Troubleshooting

Using interactive python

You can use python in the interactive mode to make tests with the compiled classes and look for specific message parameters. Open a command line in your workstation, type in python to enter the interactive mode, and use commands similar to Figure 12:



```
C:\WINDOWS\system32\cmd.exe - python
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> import sci_message_pb2
>>> with open('6848.bin','rb') as f:
...     gpb_data = sci_message_pb2.SciMessage().FromString(f.read())
...
>>> print (gpb_data.apClient.ap)
0C:F4:D5:19:00:A0
>>>
>>> print (gpb_data.apClient.clients[0].clientMac)
b8:63:4d:a8:c5:23
>>>
```

FIGURE 12: PYTHON IN INTERACTIVE MODE

MQTT connection error messages

OpenSSL Error:error:1408F119:SSL routines:SSL3_GET_RECORD:decryption failed or bad record mac
Reason: incorrect PSK key or unsupported mosquitto broker

OpenSSL Error:error:1408B0DF:SSL routines:SSL3_GET_CLIENT_KEY_EXCHANGE:psk identity not found
Reason: user not found in PSK file

OpenSSL Error:error:1408A10B:SSL routines:SSL3_GET_CLIENT_HELLO:wrong version number
Reason: incorrect TLS version

mosquitto_sub to decode bytes to hexadecimal

The `mosquitto_sub` software that comes with version 1.3.5 cannot decode the received bytes to hexadecimal, but later versions can. Here are the results of running `mosquitto_sub` without decoding:

```
C:\WINDOWS\system32\cmd.exe - mosquito_sub -t #
"test"
"test"
00
0
00 11 00
00 'uSZ-E-SalesDemo 3.0 5.1.2.0
0 " 0s13 lbstest
"test"
"test"
"test"
"test"
♥♥ T] :Y ♥^$t>+†+Ξ0
00â1LûNîR-t||1s uññRli=yP|i±uuRñ¿iR||
"test"
"test"
"test"
"test"
♥♥ T] :m ♥^t>+†+Ξ0
00â1LNNîR-t||1sâqñRli=yP|p±ñ±RÇfçR||
"test"
"test"
00â1Lñâ¥£-f||1s Lñ||£li=yP| L±L||£p¼-û£†Y3■£ñL°âÇ£Ll¿4k[1p°6âRxB♥ñppfââR||
"test"
"test"
00â1Lâñf££-f||1s Lñ||£li=yP| Lñ||=£p¼ññ£L°âMip»|¬£†Y3■£ñL°â¥£xB♥ñppâ¥££||
```

FIGURE 13 – MQTT MESSAGES WITHOUT DECODING

Download and install mosquitto version 1.5.6 in a different directory and run the following command:

```
mosquitto_sub -t # -F %I\0%l\0%t\0%x
```

The resulting readings will show the timestamp, packet length, topic and payload in hexadecimal values:

```
C:\WINDOWS\system32\cmd.exe - mosquito_sub -t # -F %I,%O,%t,%O%x

C:\Program Files (x86)\mosquitto\156>mosquitto_sub -t # -F %I,%O,%t,%O%x

2019-12-18T11:47:36-0300 99 3.0/L0C/lb1stest/743E2B1805F0/PAR 0403005f5dfa3c1100035e2f743e2b1805f0020900071062e5fe3e8370a8
e78e103dfe70709fa1a19ecc0dec678e31c0a7a6a19e1c12b0ba3173c0bebcb9c9e869cd7950b4c0cd0cf9e

2019-12-18T11:47:48-0300 6 "test" 227465737422

2019-12-18T11:47:48-0300 6 "test" 227465737422

2019-12-18T11:47:48-0300 6 "test" 227465737422

2019-12-18T11:47:53-0300 6 "test" 227465737422

2019-12-18T11:47:56-0300 99 3.0/L0C/lb1stest/743E2B1805F0/PAR 0403005f5dfa3c2500035e2c743e2b1805f0020700760d2482e8e0370a8
ecc0dec678e31c0a65a19d1c12b0ba317310b8b7b39dc869cd7950b4c0cecfce9da03be3066b3870a2a5a09e

2019-12-18T11:48:16-0300 99 3.0/L0C/lb1stest/743E2B1805F0/PAR 0403005f5dfa3c3900035e2d743e2b1805f002090007185933dba7d2c0a4
dc0dec678e31c0a5a5a09d1c12b0ba3173c0bcbcb89dc869cd7950b4c0d0d0cf9d149182b5cab6c0a3a5a09d

2019-12-18T11:48:19-0300 6 "test" 227465737422

2019-12-18T11:48:19-0300 6 "test" 227465737422

2019-12-18T11:48:19-0300 6 "test" 227465737422

2019-12-18T11:48:25-0300 6 "test" 227465737422

2019-12-18T11:48:36-0300 99 3.0/L0C/lb1stest/743E2B1805F0/PAR 0403005f5dfa3c400035e2e743e2b1805f0020900076c94f8e84d8d70af
dcc0dec678e31c0a7a89f9d1c12b0ba3173c0bdcbb9d4c74bfcd0634c09ea29e9dc869cd7950b4c0d0d0d09d

2019-12-18T11:48:51-0300 6 "test" 227465737422

2019-12-18T11:48:51-0300 6 "test" 227465737422

2019-12-18T11:48:51-0300 6 "test" 227465737422

2019-12-18T11:48:56-0300 99 3.0/L0C/lb1stest/743E2B1805F0/PAR 0403005f5dfa3c6100035e2f743e2b1805f0020a00071062e5fe3e8370ab
dcc0dec678e31c0a5a6a09e1c12b0ba3173c0bebcb9c9e869cd7950b470cfdd09ea03be3066b3870a3a3a29e

2019-12-18T11:48:57-0300 6 "test" 227465737422
```

FIGURE 14 – MQTT MESSAGES WITH DECODING

References

Eclipse Mosquitto - <https://mosquitto.org/>

Protocol Buffers Tutorial - <https://developers.google.com/protocol-buffers/docs/pythontutorial>

Eclipse Paho MQTT Client - <https://www.eclipse.org/paho/clients/python/>

MQTT and Python for Beginners - <http://www.steves-internet-guide.com/mqtt-python-beginners-course/>

SmartZone 5.2 Getting Started Guide on GPB/MQTT interface -

<https://support.ruckuswireless.com/documents/3144-smartzone-5-2-0-ga-getting-started-guide-on-gpb-mqtt-interface-sz100-sz300-vs2>

Summary

This document explained the SmartZone MQTT/GPB architecture and guided you through the steps to retrieve and decode the statistical data types provided by the Northbound Data Streaming service. The instructions presented in this document should be easily transferred to other programming languages and platforms.

Ruckus solutions are part of CommScope's comprehensive portfolio for Enterprise environments (indoor and outdoor).

We encourage you to visit **commscope.com** to learn more about:

- Ruckus Wi-Fi Access Points
- Ruckus ICX switches
- SYSTIMAX and NETCONNECT: Structured cabling solutions (copper and fiber)
- imVision: Automated Infrastructure Management
- Era and OneCell in-building cellular solutions
- Our extensive experience about supporting PoE and IoT

COMMSCOPE®

RUCKUS®

commscope.com

Visit our website or contact your local CommScope representative for more information.

© 2020 CommScope, Inc. All rights reserved.

Unless otherwise noted, all trademarks identified by ® or ™ are registered trademarks, respectively, of CommScope, Inc. This document is for planning purposes only and is not intended to modify or supplement any specifications or warranties relating to CommScope products or services. CommScope is committed to the highest standards of business integrity and environmental sustainability with a number of CommScope's facilities across the globe certified in accordance with international standards, including ISO9001, TL9000, ISO14001 and ISO45001. Further information regarding CommScope's commitment can be found at www.commscope.com/About-Us/Corporate-Responsibility-and-Sustainability.