ProviewR
OPEN SOURCE PROCESS CONTROL

# Release Notes V5.9

2021 10 22

## Table of Contents

Upgrading to ProviewR V5.9.0 ..... 4
New functions ..... 4
  Ge dynamic script arguments ..... 4
  Ge graph opening and closing scripts ..... 4
  Xtt command ‘set subevents’ ..... 4
  Xtt command ‘open file’ ..... 4
  Window icons color changed to orange ..... 4
  Guide to Storage Enviroment ..... 4
  Status monitor, Xtt and Runtime monitor started with ssh ..... 4
  Remote transactions with MQTT ..... 5
  MQTT IO ..... 6
  Home automation with zigbee2mqtt ..... 8
  ProviewR Mqtt server ..... 9
    Get ..... 9
    Set ..... 9
    Subscribe ..... 10
    Closesub ..... 10
    Sublist ..... 11
    Closesub ..... 11
    History ..... 12
    Eventhist ..... 12
    Python example 1 ..... 14
    Python example 2 ..... 14
    Python example 3 ..... 17
Upgrade procedure ..... 18

# Upgrading to ProviewR V5.9.0

This document describes new functions i ProviewR V5.8.0, and how to upgrade a project from V5.7.0 to V5.8.0.

# New functions

### *Ge dynamic script arguments*

Dynamics for executing scripts in Ge, DigScript and Script, can now handle script arguments.

The arguments are set in the ‘Argument’ property and is used as p1, p2 etc in the script. This can be used for example to transmit the current instance object for an object graph to the script.

### *Ge graph opening and closing scripts*

Script can be executed when a Ge graph is opened and when it’s closed. This is done with the Script action that has the new property TriggerEvent, that can be set to Open or Close. Open will execute the script when the graph is opened, and Close when the graph is closed. The default ClickMB1 will execute the script when MB1 is clicked on the object.

### *Xtt command ‘set subevents’*

The command disables events in a Ge window or table object.

Window and table objects grab all events inside their area, and click sensitive objects can’t be placed on top of them. If they temporarily should be covered by a sensitive object though, the eventhandling can be turned off for window and table objects in the graph.

```
xtt> set subevents ‘graphname’ [/on] [/off]
```

### *Xtt command ‘open file’*

Opens a file or URL in the user’s preferred application.

```
xtt> open file ‘filename’
```

### *Window icons color changed to orange*

To easier separate runtime and development windows, the runtime icons are now orange and the development icons blue.

### *Guide to Storage Enviroment*

New documentation for the storage server.

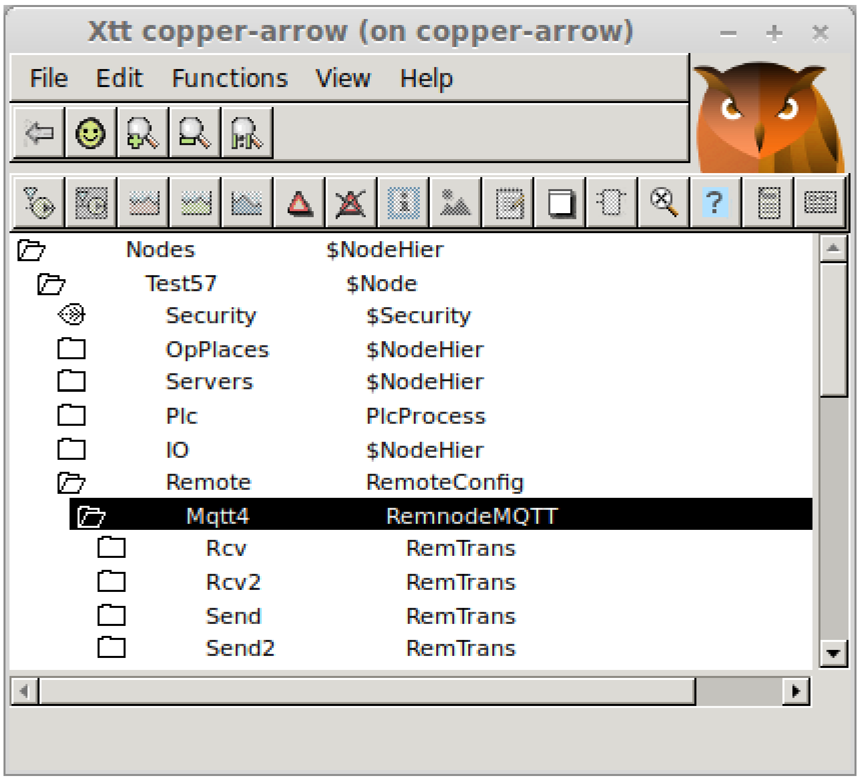
### *Status monitor, Xtt and Runtime monitor started with ssh*

Xtt and Runtime monitor was previously started via gsoap and required remote access the X server.

For security reasons ssh is used instead.

## *Remote transactions with MQTT*

Remote transactions can now be sent and received via MQTT with the RemnodeMQTT object.


Xtt copper-arrow (on copper-arrow)
File Edit Functions View Help

```
Nodes             $NodeHier
  Test57            $Node
    Security          $Security
    OpPlaces          $NodeHier
    Servers           $NodeHier
    Plc               PlcProcess
    IO                $NodeHier
    Remote            RemoteConfig
      Mqtt4             RemnodeMQTT
        Rcv               RemTrans
        Rcv2              RemTrans
        Send              RemTrans
        Send2             RemTrans
```

**Fig Remote configuration with MQTT**

The communication is configured with a RemnodeMQTT object. The MQTT server, topics for publishing and subscribing are specified here.

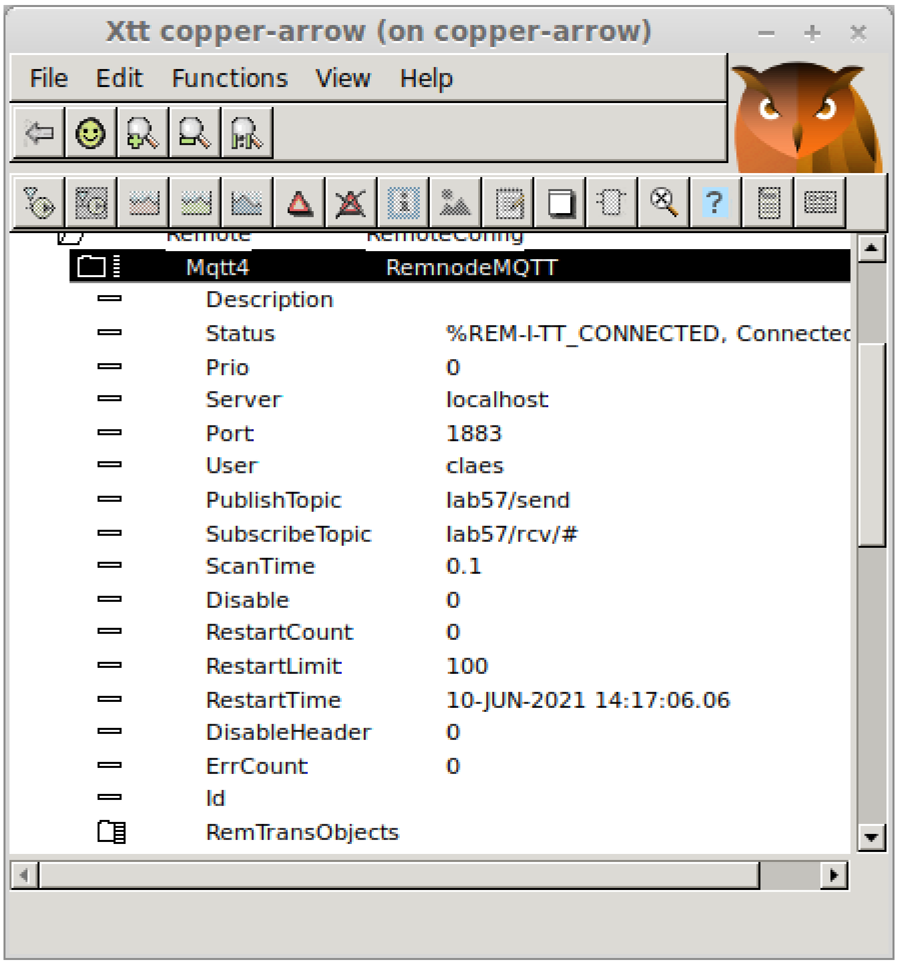
The specification of topics differs if the remote header is disabled or not.

If the header is present

- Sending: publishing is made with the  topic in PubishTopic in the RemnodeMQTT. Address[0] and Address[1] in the RemTrans object is used to match RemTrans objects.
- Receiving: Subscriptions are med with the  topic in SubscribeTopic in the RemnodeMQTT object. The message is directed to the RemTrans with matching Address[0] and Address[1].

If header is disabled

- Sending: publishing is made with the topic in RemTrans.TransName
- Receiving: A generic topic is set in SubscribeTopic in the RemnodeMQTT object, eg 'lab57/rcv/#'. A more narrow topic is set in RemTrans.TransName, eg 'lab57/rcv/msg1'.


Xtt copper-arrow (on copper-arrow)
File Edit Functions View Help

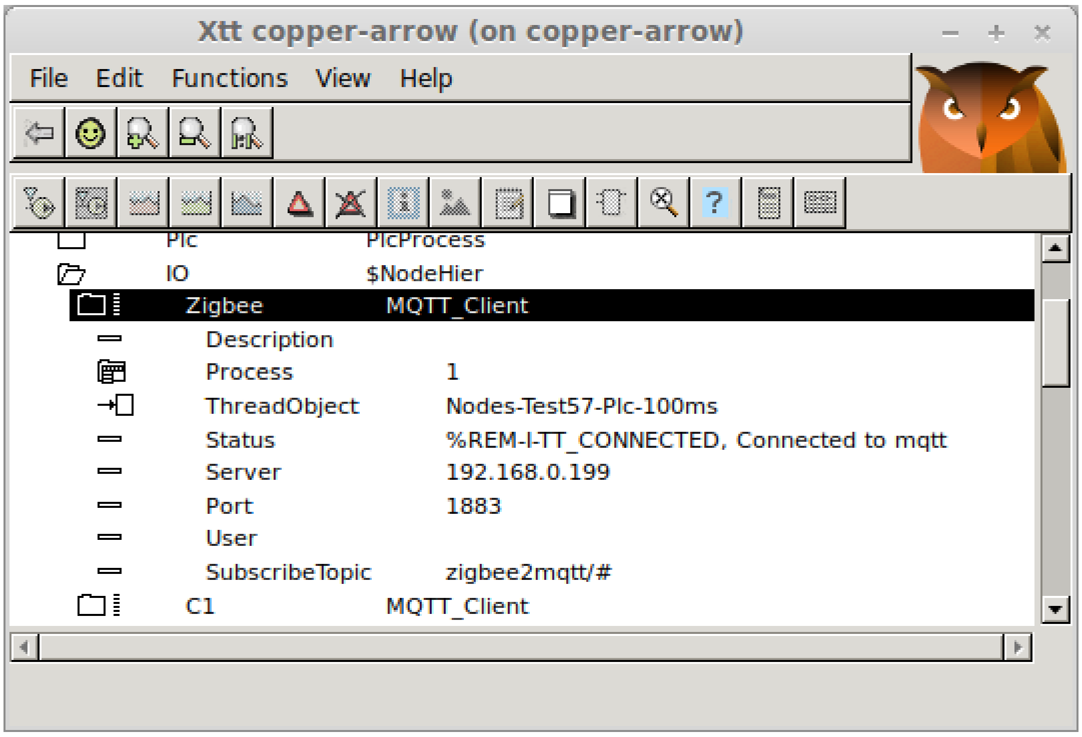
```
Remote                  RemoteConfig
Mqtt4                   RemnodeMQTT
    Description
    Status              %REM-I-TT_CONNECTED, Connectec
    Prio                0
    Server              localhost
    Port                1883
    User                claes
    PublishTopic        lab57/send
    SubscribeTopic      lab57/rcv/#
    ScanTime            0.1
    Disable             0
    RestartCount        0
    RestartLimit        100
    RestartTime         10-JUN-2021 14:17:06.06
    DisableHeader       0
    ErrCount            0
    Id
    RemTransObjects
```

**Fig RemnodeMQTT object**

## *MQTT IO*

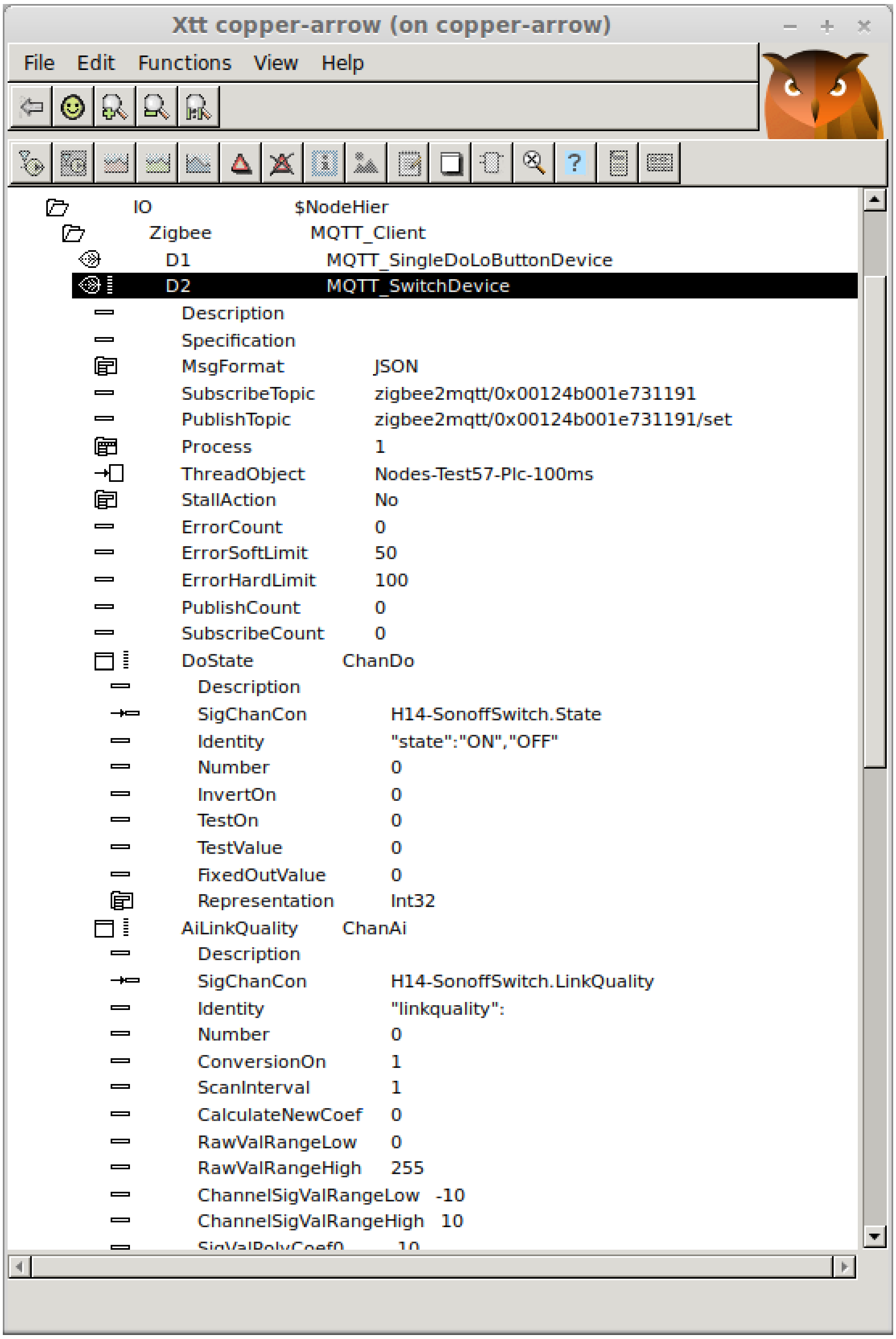
The MQTT IO can read values published on an MQTT server into insignals and publish values of outsignals to an MQTT server.

The configuration is done with a MQTT_Client object and MQTT_Device objects.


Xtt copper-arrow (on copper-arrow)
File Edit Functions View Help

```
Plc                   PlcProcess
IO                    $NodeHier
  Zigbee                MQTT_Client
    Description
    Process               1
    ThreadObject          Nodes-Test57-Plc-100ms
    Status                %REM-I-TT_CONNECTED, Connected to mqtt
    Server                192.168.0.199
    Port                  1883
    User
    SubscribeTopic        zigbee2mqtt/#
  C1                    MQTT_Client
```

**Fig MQTT_Client**


Xtt copper-arrow (on copper-arrow)
File Edit Functions View Help

```
IO                  $NodeHier
  Zigbee              MQTT_Client
    D1                  MQTT_SingleDoLoButtonDevice
    D2                  MQTT_SwitchDevice
      Description
      Specification
      MsgFormat               JSON
      SubscribeTopic          zigbee2mqtt/0x00124b001e731191
      PublishTopic            zigbee2mqtt/0x00124b001e731191/set
      Process                 1
      ThreadObject            Nodes-Test57-Plc-100ms
      StallAction             No
      ErrorCount              0
      ErrorSoftLimit          50
      ErrorHardLimit          100
      PublishCount            0
      SubscribeCount          0
      DoState             ChanDo
        Description
        SigChanCon              H14-SonoffSwitch.State
        Identity                "state":"ON","OFF"
        Number                  0
        InvertOn                0
        TestOn                  0
        TestValue               0
        FixedOutValue           0
        Representation          Int32
      AiLinkQuality       ChanAi
        Description
        SigChanCon              H14-SonoffSwitch.LinkQuality
        Identity                "linkquality":
        Number                  0
        ConversionOn            1
        ScanInterval            1
        CalculateNewCoef        0
        RawValRangeLow          0
        RawValRangeHigh         255
        ChannelSigValRangeLow   -10
        ChannelSigValRangeHigh  10
        SigValPolyCoef0         10
```

**Fig Switch object based on MQTT_Device with a Do and an Ai channel**

## *Home automation with zigbee2mqtt*

For those interested in home automation I just want to mention zigbee2mqtt that makes it possible to access a large range of zigbee devices. Zigbee2mqtt is flashed to a usbstick that acts a zigbee coordinator. A server program is run on RasberryPi that presents the zigbee communication on an

MQTT server in json format. This can then be accessed by the ProviewR MQTT IO.

## *ProviewR Mqtt server*

The ProviewR Mqtt server is a program that can answer request via MQTT about attribute values and history data. It is also possible to setup subscriptions and set values.

The server is configured with a MqttServer object in the node hierarchy. When this object is created the server program rt_mqtt_server will start.

Requests can be put to a specific topic on an MQTT broker. The topic is configured in the MqttServer object and is 'proviewr/server' by default. The request should be on json format and contain "action" that specifies the type of request. The action can be get, set, subscribe, closesub, sublist, closesublist or history.

### Get

Get the value of an attribute.

**Request**

| | |
|---|---|
| action | Should be "get". |
| attribute | Full name of the attribute. |
| reply | MQTT topic where the reply should be published. |

**Reply**

| | |
|---|---|
| status | Return status. |
| value | Attribute value. |

**Example**

Request

{"action":"get","attribute":"H16-Av1.ActualValue","reply":"repl/get"}

Reply

{"status":141459465,"value":45.9577}

### Set

Set the value of an attribute.

**Request**

| | |
|---|---|
| action | Should be "set". |
| attribute | Full name of the attribute. |
| reply | MQTT topic where the reply should be published. |

**Reply**

| | |
|---|---|
| status | Return status. |

**Example**

Request

{"action":"set","attribute":"H16-Av1.ActualValue","reply":"repl/get"}

Reply

{“status”:141459465}

## Subscribe

Set up a subscription of an attribute. The value of the attribute should cyclically be sent to the reply topic until the duration time has elapsed, or the subscription is closed with a closesub message.

**Request**

| | |
|---|---|
| action | Should be “subscribe”. |
| attribute | Full name of the attribute. |
| cycle | Cycle time in seconds. |
| duration | Max duration of the subscription. |
| reply | MQTT topic where the reply should be published. |

**Reply**

| | |
|---|---|
| subref | Subscription reference. |
| status | Not yet implemented. |
| value | Attribute value. |

**Example**

Request

{"action":"subscribe","attribute":"H16-Av1.ActualValue","cycle":"1.0","duration":"300.0","reply":"repl/subscribe"}

Reply

{'subref': 1, 'status': 0, 'value': -14.98953}

## Closesub

Close a subscription started with the subscribe action.

**Request**

| | |
|---|---|
| action | Should be “closesub”. |
| subref | Subscription reference. |

No reply is sent.

**Example**

Request

{"action":"closesub","subref”:5}

## Sublist

Set up a subscription of a number of attributes. The values of the attributes should cyclically be sent to the reply topic until the duration time has elapsed, or the subscription is closed with a closesublist message.

**Request**

| | |
|---|---|
| action | Should be “sublist”. |
| attribute | Array of index and full name of the attribute. |
| cycle | Cycle time in seconds. |
| duration | Max duration of the subscription. |
| reply | MQTT topic where the reply should be published. |

**Reply**

| | |
|---|---|
| subref | Subscription reference. |
| a | Array of index and attribute value. |

**Example**

Request

{"action":"sublist","cycle":"1.0","duration":"15.0","reply":"repl/sublist","attribute":[{1,"H16-Av1.ActualValue"},{2,"H16-Av2.ActualValue"},{3,"H16-Av4.ActualValue"}]}

Reply

{"subref":2,"a":[{"idx":1,"value":39.3397},{"idx":2,"value":39.3397},{"idx":3,"value":42.1814}]}

## Closesub

Close a subscription started with the subscribe action.

**Request**

| | |
|---|---|
| action | Should be “closesublist”. |
| subref | Subscription reference. |

No reply is sent.

**Example**

Request

{"action":"closesublist","subref”:2}

## History

Get process history for an attribute.

Returns process history from a sev database.

**Request**

| | |
|---|---|
| action | Should be “history”. |
| server | Server from which the history should be fetched. |
| object | Full object name. |
| attribute | Attribute name. |
| from | Start time for history data, eg '20-JUN-2021 12:00:00'. |
| to | End time for history data, eg '21-JUN-2021 12:00:00'.<br>To get the most recent data 'now' can be used for the current time, and a delta time can be given in 'from', eg 'to':'now' and 'from':'1 00:00:00'. |
| maxpoints | Max number of points that should be returned. |
| reply | MQTT topic where the reply should be published. |

**Reply**

| | |
|---|---|
| status | Status of the action. |
| values | Array of attribute values. |
| time | Array of time for the values. |

**Example**

Request

{"action":"history","server":"localhost","object":"H1-Av1",”attribute”:”ActualValue”"reply":"repl/history","from":”0:15:0”, “to”:”now”,”maxrows”:6}

Reply

{“status":1315888905,"values":[-96.1745, -34.2858, 82.5293, 99.5304, 96.0948, 56.3516],”time”:["14-JUN-2021 15:08:48.00","14-JUN-2021 15:09:09.00","14-JUN-2021 15:09:17.00","14-JUN-2021 15:09:23.00","14-JUN-2021 15:09:38.00","14-JUN-2021 15:09:59.00"]}

## Eventhist

Get alarm and event history for a table defined by a SevHistEvent object.

Returns event history from a sev database.

**Request**

| | |
|---|---|
| action | Should be “eventhist”. |
| server | Server from which the history should be fetched. |
| object | Name or identity of SevHistEvent object. |
| eventtype | Mask for event types that should be searched for. Optional.<br>1: Ack,<br>2: Block |

| | |
|---|---|
| | 4: Cancel<br>8: CancelBlock<br>16: Missing<br>32: Reblock<br>64: Return<br>128: Unblock<br>256: InfoSuccess<br>512: Alarm<br>1024: MaintenanceAlarm<br>2048: SystemAlarm<br>4096: UserAlarm1<br>8192: UserAlarm2<br>16384: UserAlarm3<br>32768: UserAlarm4<br>65536: Info |
| eventprio | Mask for event priorities that should be searched for. Optional.<br>1: Prio A<br>2: Prio B<br>4: Prio C<br>8: Prio D |
| text | Event text with wild card that should be search for. Optional. |
| name | Event name with wild card that should be searched for. Optional. |
| from | Start time for history data, eg '20-JUN-2021 12:00:00'. |
| to | End time for history data, eg '21-JUN-2021 12:00:00'.<br>To get the most recent data 'now' can be used for the current time, and a delta time can be given in 'from', eg 'to':'now' and 'from':'1 00:00:00'. |
| options | Mask that states which values should be returned. Optional.<br>1: Time<br>2: Event type<br>4: Event priority<br>8: Event text<br>16: Event name<br>32: Event identity |
| maxpoints | Max number of points that should be returned. |
| reply | MQTT topic where the reply should be published. |

**Reply**

| | |
|---|---|
| status | Status of the action. |
| time | Array of event time. |
| type | Array of event type. |
| prio | Array of event priority. |
| text | Array of event text. |
| name | Array of event name. |
| id_nix | Array of event identity, nix. |
| id_idx | Array of event identity, idx. |

**Example**

Request

{"action":"eventhist","server":"localhost","object":"H1-SevHistEvents","reply":"repl/eventhist","from":”7 0:0:0”, “to”:”now”,”maxrows”:10}

Reply

{"status":135888905,"time":["07-JUN-2021 15:06:21.00","07-JUN-2021 15:06:55.00","07-JUN-2021 15:11:56.00","07-JUN-2021 15:11:56.00","07-JUN-2021 15:11:56.00","07-JUN-2021 15:11:56.00","07-JUN-2021 15:11:58.00","07-JUN-2021 15:11:58.00","07-JUN-2021 15:12:02.00","07-JUN-2021 15:12:02.00"],"type":[256,7,64,64,64,64,64,64,7,7],"text":["System status error, node copper-arrow","","Dv 8 alarm","Dv 7 alarm","Dv 2 alarm","Dv 1 alarm","Dv 10 alarm","Dv 9 alarm","Dv 10 return","Dv 9 return"]}

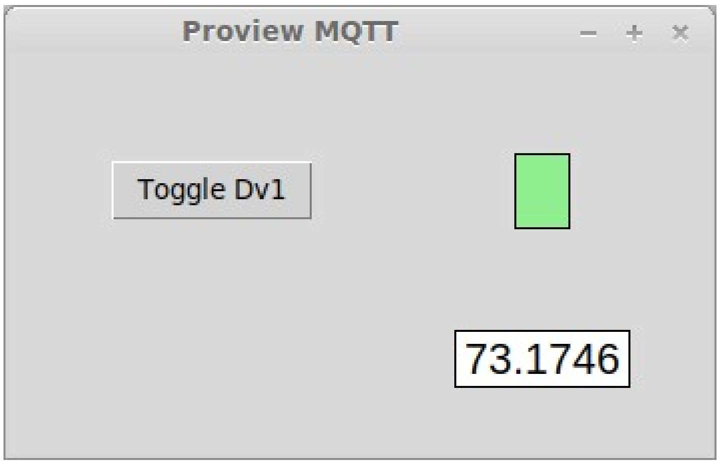
## Python example 1

This example fetches the value for an attribute with the 'get' action.

```
#!/usr/bin/python3
#
import paho.mqtt.client as mqtt
import sys
import time
from datetime import datetime
import json

# Print reply
def on_message(client, userdata, message):
    reply = json.loads(str(message.payload.decode("utf-8")))
    value = reply['value']
    print("Reply:", reply)
    print("Value:", value)

# Connect to MQTT on localhost
client = mqtt.Client('MyClient')
client.username_pw_set('pwrp','pwrp')
client.on_message = on_message
client.connect('localhost')

# Send request and subscribe on reply
client.subscribe("repl/get", 1);
client.publish('proviewr/server', '{"action":"get","attribute":"H16-
Av1.ActualValue","reply":"repl/get"}')

# Wait for reply
for i in range (0, 3):
  client.loop_start()
  time.sleep(1)
  client.loop_stop()
```

## Python example 2

Simple  graph with a pushbutton, an indicator and a value field.

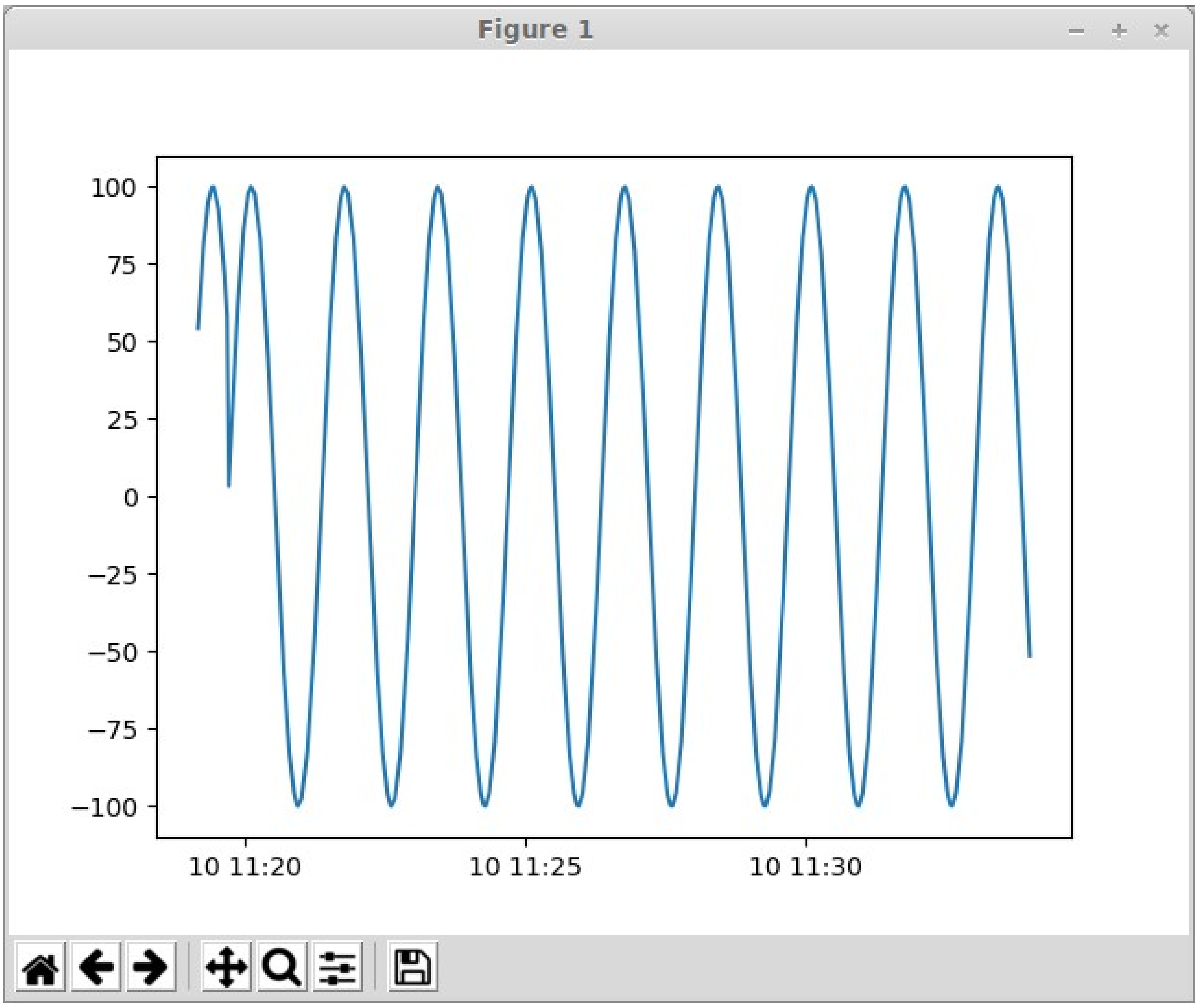


```
#!/usr/bin/python3
#
from tkinter import *
import paho.mqtt.client as mqtt
import sys
import time
from datetime import datetime
import json
import random

def on_closing():
    global subref

    # Close subsciptions
    if subref != 0:
        client.publish('proviewr/server',
'{"action":"closesublist","subref":' + str(subref) + '}')

    window.destroy()


def on_message(client, userdata, message):
    global val1
    global val2
    global subref
    global set_reply
    global sublist_reply

    if message.topic == sublist_reply:
        data = json.loads(str(message.payload.decode("utf-8")))
        subref = data['subref']
        val1 = data['a'][0]['value']
        val2 = data['a'][1]['value']

    if message.topic == set_reply:
        pass

# Button click callback
def button_click_cb():
    global set_reply

    client.subscribe(set_reply, 1);
    if val2 == 0:
        client.publish('proviewr/server', '{"action":"set","attribute":"H17-
Dv1.ActualValue","value":"1","reply":"' + set_reply + '"}')
    else:
        client.publish('proviewr/server', '{"action":"set","attribute":"H17-
Dv1.ActualValue","value":"0","reply":"' + set_reply + '"}')

# Cyclic scan function
def scan():
    global sub1_old
    global sub2_old
    global val1
    global val2

    if val2 != sub2_old:
        if val2 == 1:
            dv1_label["bg"] = "lightgreen"
        else:
            dv1_label["bg"] = "black"
        sub2_old = val2

    if val1 != sub1_old:
        av1_label["text"] = val1
        sub1_old = val1

    window.after(500, scan)

# Create window
window = Tk()
window.title("Proview MQTT")
window.geometry('350x200')

# Create unique name and topics to be able to run serveral instances
rand = str(random.randint(1,999999))
name = 'MqttTest' + rand
sublist_reply = 'repl/' + rand + '/sublist'
set_reply = 'repl/' + rand + '/set'

# Create button
button = Button(window, text="Toggle Dv1", command=button_click_cb,
bg="lightgray")
button.grid(column=0, row=0, padx=50, pady=50)

# Create indicator label
dv1_label = Label(window, width=3, height=2, bg="black", borderwidth=1,
relief="solid")
dv1_label.grid(column=1, row=0, padx=50, pady=50)

# Create value label
av1_label = Label(window, width=7, bg="white", borderwidth=1,
relief="solid",
font=("Helvetica",16))
av1_label.grid(column=1, row=2, padx=20, pady=0)

# Attach MQTT
client = mqtt.Client(name)
client.username_pw_set('pwrp','pwrp')
client.on_message = on_message
client.connect('localhost')
```

```
# Set up subscriptions
client.subscribe(sublist_reply, 1);
client.publish('proviewr/server',
'{"action":"sublist","cycle":"1.0","duration":"150.0","reply":"' +
sublist_reply + '","attribute":[{1,"H17-Av1.ActualValue"},{2,"H17-
Dv1.ActualValue"}]}')

subref = 0
sub1_old = -1
sub2_old = -1
val1 = 0
val2 = 0

window.protocol("WM_DELETE_WINDOW", on_closing);

client.loop_start()
scan()
window.mainloop()
```

## Python example 3

Drawing a history curve with matplotlib



```
#!/usr/bin/python3
#
import paho.mqtt.client as mqtt
import sys
import time
from datetime import datetime
import json
import matplotlib.pyplot as plt
from datetime import datetime

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

def on_message(client, userdata, message):
    print("message received ", datetime.now(),
str(message.payload.decode("utf-8")), flush=True)
    data = json.loads(str(message.payload.decode("utf-8")))

    # Convert time strings to datetime objects
    t = []
    for dt in data['time']:
        t.append(datetime.strptime(dt+'0000', '%d-%b-%Y %H:%M:%S.%f'))

    # Plot the curve, use drawstyle='steps-pre' for digital signals
    plt.plot(t, data['values'], label='Diff')
    plt.show()


# Connect to MQTT server
client = mqtt.Client('Claes')
client.username_pw_set('pwrp','pwrp')
client.on_message = on_message
client.connect('localhost')

# Subscribe to reply
client.subscribe("repl/history", 1)

# Send history request
client.publish('proviewr/server',
'{"action":"history","reply":"repl/history","server":"localhost","object":"H
1-
Av1","attribute":"ActualValue","from":"0:15:0","to":"now","maxrows":2000}')

for i in range (0, 3):
  print("Loop");
  client.loop_start()
  time.sleep(1)
  client.loop_stop()
```

# Upgrade procedure

The upgrading has to be done from any V5.8. If the project has a lower version, the upgrade has to be performed stepwise following the schema

V2.1 -> V2.7b -> V3.3 -> V3.4b -> V4.0.0 -> V4.1.3 ->V4.2.0->V4.5.0->V4.6.0->V4.7.0->V4.8.6->(V5.0.0)->V5.1.0->V5.2.0->V5.3->V5.4->V5.5->V5.6->V5.7->V5.8->V5.9

Enter the administrator and change the version of the project to V5.9.0. Save and close the administrator.

Enter the directory volume and save.

I you have any class volumes, enter the class editor and build the volume.

Enter the configurator for each root volume and activate 'Function/Update Classes' and build.