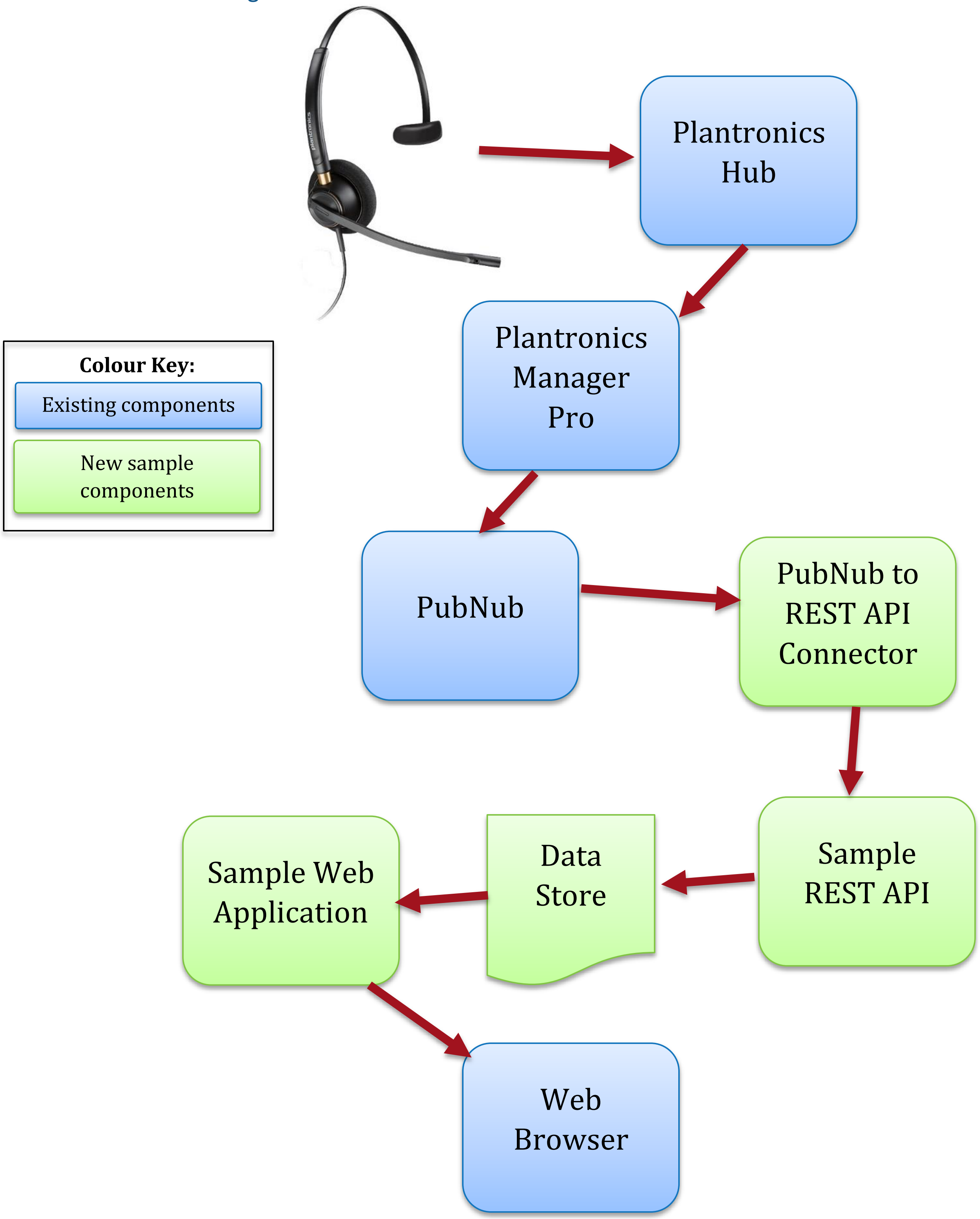
Plantronics Web APIs – Realtime PubNub to REST API Connector Sample

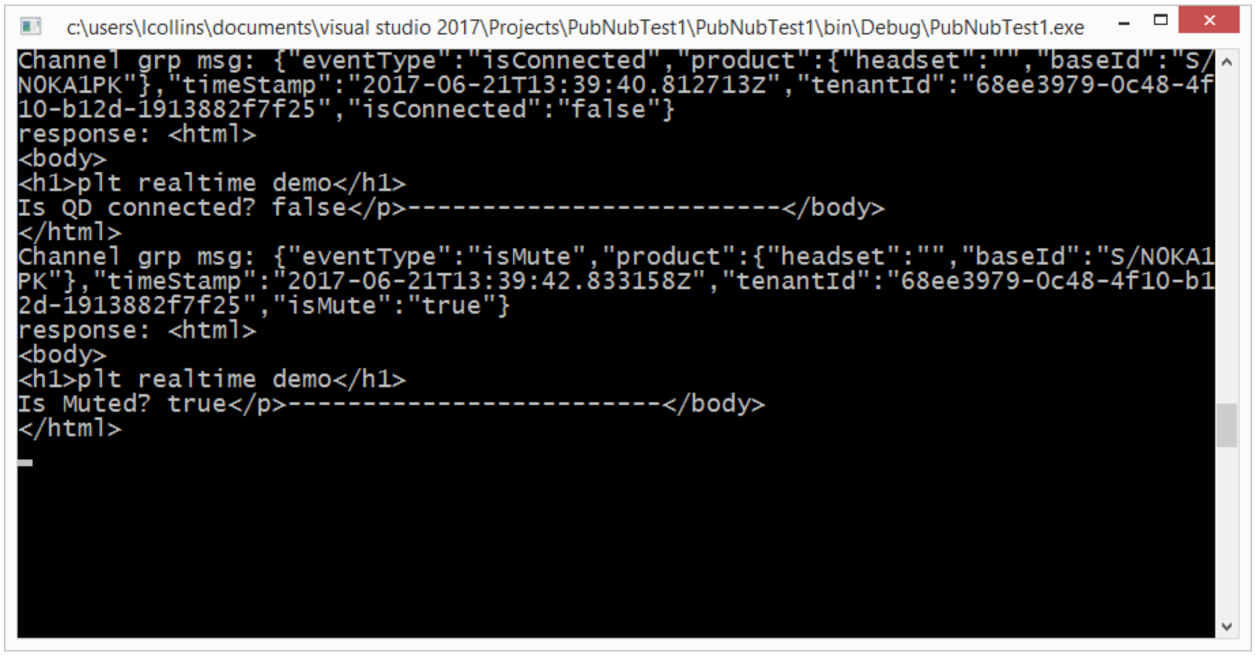
Author: Lewis Collins, Plantronics, 21st June 2017

Overview Diagram



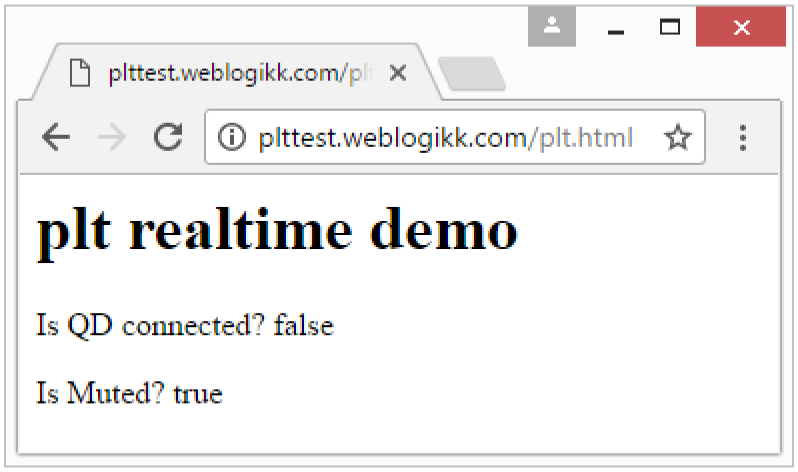
The solution in action

Below is screenshot of the “PubNub to REST API Connector” program (PubNubTest1.exe) running on the console. Each time a pubnub event is received it reposts it to the “Sample REST API” PHP script. (Note: some debug in the form of HTML tags is returned by the Sample REST API script to show it is working).


c:\users\lcollins\documents\visual studio 2017\Projects\PubNubTest1\PubNubTest1\bin\Debug\PubNubTest1.exe

```
Channel grp msg: {"eventType":"isConnected","product":{"headset":"","baseId":"S/
NOKA1PK"},"timeStamp":"2017-06-21T13:39:40.812713Z","tenantId":"68ee3979-0c48-4f
10-b12d-1913882f7f25","isConnected":"false"}
response: <html>
<body>
<h1>plt realtime demo</h1>
Is QD connected? false</p>-------------------------</body>
</html>
Channel grp msg: {"eventType":"isMute","product":{"headset":"","baseId":"S/NOKA1
PK"},"timeStamp":"2017-06-21T13:39:42.833158Z","tenantId":"68ee3979-0c48-4f10-b1
2d-1913882f7f25","isMute":"true"}
response: <html>
<body>
<h1>plt realtime demo</h1>
Is Muted? true</p>-------------------------</body>
</html>
```

Below is a browser window that is displaying the current QD and mute state of the user’s DA90 headset. This sample app is hosted on Amazon EC2 and can therefore be viewed anywhere in the world.



plt realtime demo

Is QD connected? false

Is Muted? true

New Sample Component Summary

PubNub to REST API Connector

This program is written in C# and uses a PubNub library from NuGet called “Pubnub” to subscribe to and listen for the realtime QD and mute events from the Plantronics Web Services stream API.

When it receives one forwards it to the Sample REST API using another NuGet library called “Microsoft.AspNet.WebApi.Client”.

I have highlighted below the PubNub subscribe and publish keys, the PubNub channel id and the Uri and script name of the Sample REST API script.

PubNubTest1.sln / Program.cs

```
using System;
using PubnubApi;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;

namespace PubNubTest1
{
    class Program
    {
        static public Pubnub pubnub;

        static HttpClient client = new HttpClient();

        static public void Main()
        {
            PNConfiguration config = new PNConfiguration();
            config.SubscribeKey = "sub-c-e0e48298-c1a7-11e6-8036-0619f8945a4f";
            config.PublishKey = "pub-c-36c67bb6-e1b8-4fc9-8017-040c9cc8c4ce";

            pubnub = new Pubnub(config);
            pubnub.AddListener(new SubscribeCallbackExt(
                (pubnubObj, message) => {
            // Handle new message stored in message.Message
            if (message != null)
                    {
                        if (message.Channel != null)
                        {
                            // Message has been received on channel group stored in
                            // message.Channel()
                            System.Console.WriteLine("Channel grp msg: 
"+message.Message.ToString());
                        }
                        else
                        {
                            // Message has been received on channel stored in
                            // message.Subscription()
                            System.Console.WriteLine("Channel sub msg: " + 
message.Message.ToString());
                        }

                        ForwardMessageToHTTPService(message.Message);
                        /*
```

```
                            log the following items with your favorite logger
                                - message.Message()
                                - message.Subscription()
                                - message.Timetoken()
                        */
                    }
                },
                (pubnubObj, presence) => { },
                (pubnubObj, status) => {
                    if (status.Category ==
PNStatusCategory.PNUnexpectedDisconnectCategory)
                    {
                // This event happens when radio / connectivity is lost
            }
                    else if (status.Category == PNStatusCategory.PNConnectedCategory)
                    {
                // Connect event. You can do stuff like publish, and know you'll get it.
                // Or just use the connected event to confirm you are subscribed for
                // UI / internal notifications, etc

                pubnub.Publish()
                            .Channel("f343e170-5f4e-43c2-bcbc-4dd5e7d0a99a_sub1")
                            .Message("hello!!")
                            .Async(new PNPublishResultExt((publishResult, publishStatus)
=> {
                        // Check whether request successfully completed or not.
                        if (!publishStatus.Error)
                                {
                            // Message successfully published to specified channel.
                        }
                                else
                                {
                            // Request processing failed.

                            // Handle message publish error. Check 'Category' property to
find out possible issue
                            // because of which request did fail.
                        }
                            }));
                    }
                    else if (status.Category == PNStatusCategory.PNReconnectedCategory)
                    {
                // Happens as part of our regular operation. This event happens when
                // radio / connectivity is lost, then regained.
            }
                    else if (status.Category ==
PNStatusCategory.PNDecryptionErrorCategory)
                    {
                // Handle messsage decryption error. Probably client configured to
                // encrypt messages and on live data feed it received plain text.
            }
                }
            ));

            pubnub.Subscribe<string>()
                .Channels(new string[] {
            "f343e170-5f4e-43c2-bcbc-4dd5e7d0a99a_sub1"
                })
                .Execute();

            RunAsync().Wait();
        }

        static async Task RunAsync()
        {
            // New code:
            //client.BaseAddress = new Uri("http://localhost:8888/");
            client.BaseAddress = new Uri("http://plttest.weblogikk.com/");
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));

            System.Console.WriteLine("press enter to quit...");
            System.Console.ReadLine();
        }

        static async Task ForwardMessageToHTTPService(object message)
        {
            try
            {
                HttpResponseMessage response = await client.PostAsJsonAsync("plt.php",
message);
                response.EnsureSuccessStatusCode();

                Console.WriteLine("response: " + await
response.Content.ReadAsStringAsync());
            }
            catch(Exception e)
            {
                Console.WriteLine("error: " + e.ToString());
            }
        }
    }
}
```

Sample REST API

The Sample REST API script is written in PHP and receives the JSON from the Plantronics realtime pubnub channel that was reposted by the connector program above.

I have highlighted below how the script gets the POST data containing the JSON and how it writes the QD and Mute states to a data store (.txt files in this sample).

plt.php

```
<html>
<body>
<h1>plt realtime demo</h1>
<?php
function isValidJSON($str) {
   json_decode($str);
   return json_last_error() == JSON_ERROR_NONE;
}

//print "<p>raw data:" . $HTTP_RAW_POST_DATA . "</p><p>";
```

```
$json_params = file_get_contents("php://input");

if (strlen($json_params) > 0 && isValidJSON($json_params))
{
    //print "<br\>JSON params = ".$json_params;
    $json_params = trim(stripslashes(html_entity_decode($json_params)),'"');
    //print "<br\>JSON params 2 = ".$json_params;
    $decoded_params = json_decode($json_params, TRUE);
    //print "JSON params: " . $json_params;
    if ($decoded_params==NULL) print "WAS NULL";
    //print "JSON dump: " . var_dump($decoded_params);
    //print "Event Type = ".$decoded_params["eventType"];
    switch ($decoded_params["eventType"])
    {
        case "isConnected":
            print "Is QD connected? ".$decoded_params["isConnected"];
            file_put_contents("qdstate.txt","Is QD connected?
".$decoded_params["isConnected"]);
            break;
        case "isMute":
            print "Is Muted? ".$decoded_params["isMute"];
            file_put_contents("mutestate.txt","Is Muted? ".$decoded_params["isMute"]);
            break;
    }
}

print "</p>-------------------------";
?>
</body>
</html>
```

Data Store

qdstate.txt / mutestate.txt

The data store used in this example is a pair of plain .txt files as highlighted below.

```
Is QD connected? true
```

```
Is Muted? false
```

Sample Web Application

In order to visualize the current QD and mute state I have created a web page that polls for the current states every 500ms.

I have highlighted below how the script uses AJAX to load these states using another PHP script called “plt_getstates.php”, this second PHP script loads the current states from the data store (.txt files). This data is then inserted into a div section of the web page.

plt.html

```
<html>
<body onload="setInterval(function(){ UpdateDeviceStates() }, 500);">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
<script>
function UpdateDeviceStates()
{
    var txt = "ajax failed";

    $.ajax({url: "plt_getstates.php", async: false, success: function(result){
                txt = result;
        }});

    document.getElementById('device_states').innerHTML = txt;
}
</script>
<h1>plt realtime demo</h1>
<div id="device_states">
</div>
</html>
```

plt_getstates.php

```
<?php
$qd_state = file_get_contents("qdstate.txt");
$mute_state = file_get_contents("mutestate.txt");
print "<p>".$qd_state."</p>";
print "<p>".$mute_state."</p>";
?>
```