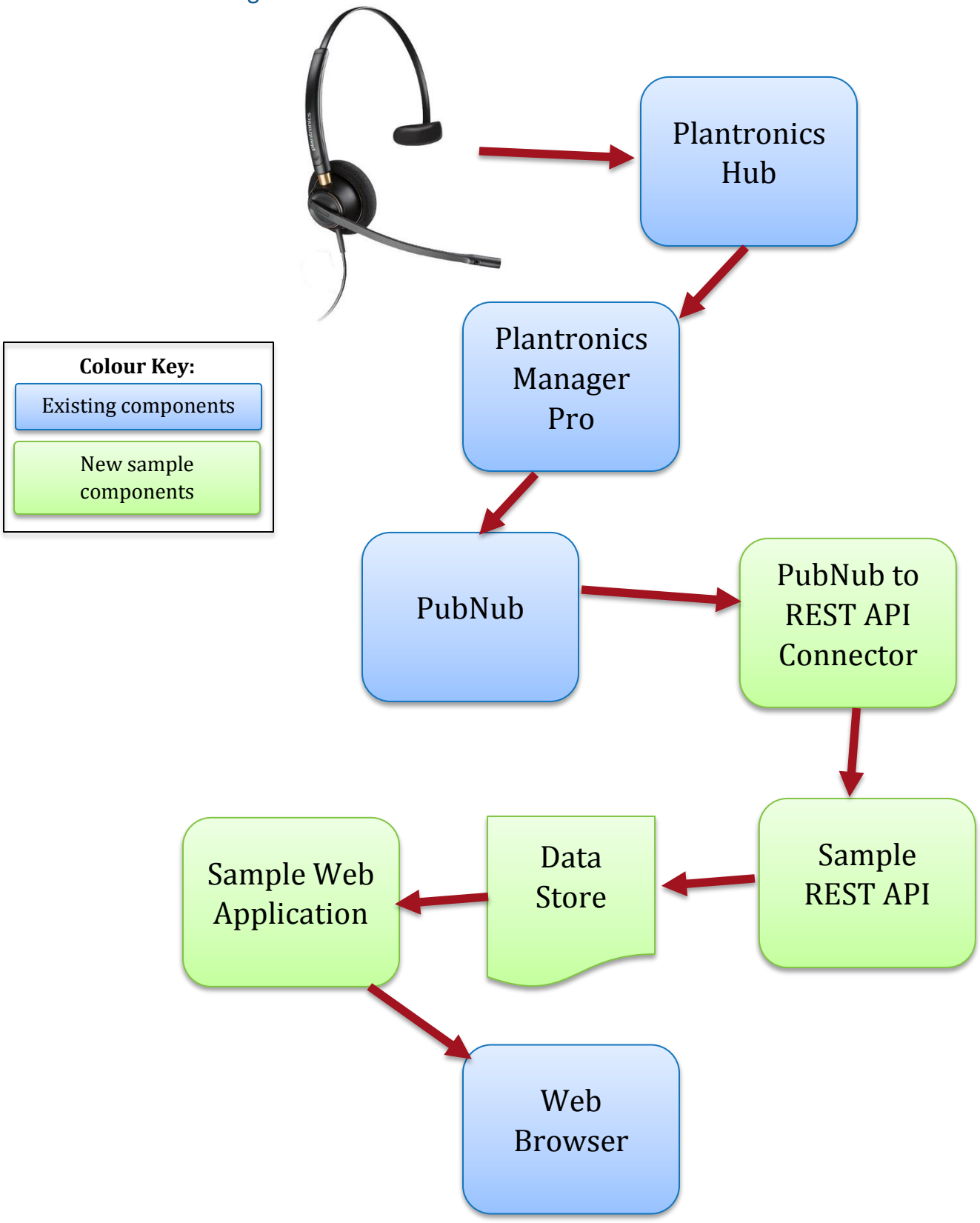
Plantronics Web APIs – Realtime PubNub to REST API Connector Sample

Author: Lewis Collins, Plantronics, 22nd June 2017

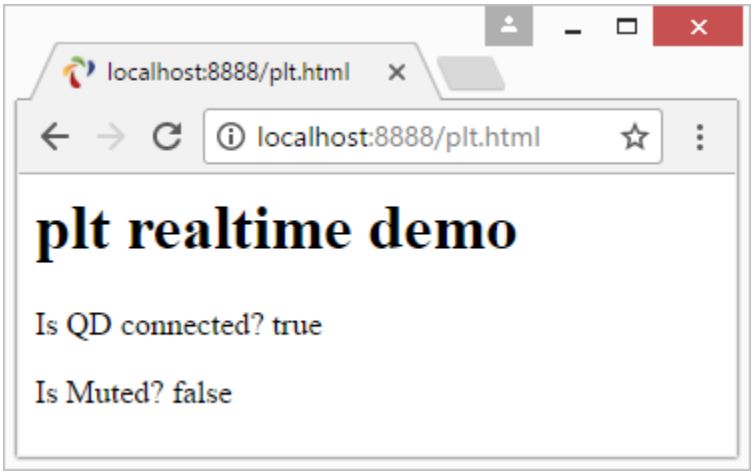
Overview Diagram



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).

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 "ampss" running on localhost but could also use a remote host.



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;
using System.Configuration;
using System.Collections.Specialized;

namespace PubNubTest1
{
    /// <summary>
    /// This Program demo illustrates how to consume the Plantronics Web Services API
realtime stream.
    /// It also illustrates how to forward those events (as JSON) onto another REST
Service API for
    /// example to connect to another system.
    /// The events it will look for are QD (quick disconnect) connected/disconnected and
Mute (muted/unmuted)
    /// NOTE: In order to work you must edit the .config file in this project (or
alongside the EXE) and add
    /// your settings, as follows:
    /// - PubNub_SubscribeKey - this key is provided by your PubNub developer account
    /// - PubNub_PublishKey - this key is provided by your PubNub developer account
    /// - PubNub_SubscribeChannel - this channel id you create in your PubNub developer
account
    /// - REST_API_BaseAddress - this is the base URL of a REST Service API to forward
the event to,
    ///     e.g. http://localhost:8888/ (can also be a remote URL)
    /// - REST_API_RequestUri - this is the script name of the REST Service API to
forward the event to,
    ///     i.e. the bit of the URL after the BaseURL, e.g. myscript.php
    /// Author: Lewis.Collins@Plantronics.com, 22nd June 2017
    /// </summary>
    class Program
    {
        static public Pubnub pubnub;

        static HttpClient client = new HttpClient();

        static public void Main()
        {
            PNConfiguration config = new PNConfiguration();
```

```
            config.SubscribeKey =
ConfigurationManager.AppSettings.Get("PubNub_SubscribeKey");
            config.PublishKey =
ConfigurationManager.AppSettings.Get("PubNub_PublishKey");

            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());
                        }

                        // Forward the message to your own REST Service API...
                        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(ConfigurationManager.AppSettings.Get("PubNub_SubscribeChannel"))
                                    .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.
                    }
                }
            ));

            // Subscripe to your PubNub channel and wait for events
            pubnub.Subscribe<string>()
                .Channels(new string[] {
                    ConfigurationManager.AppSettings.Get("PubNub_SubscribeChannel")
                })
                .Execute();

            RunAsync().Wait();
        }

        static async Task RunAsync()
        {
            // New code:
            client.BaseAddress = new
Uri(ConfigurationManager.AppSettings.Get("REST_API_BaseAddress"));
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));

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

```
        /// <summary>
        /// Forward the message to your own REST Service API...
        /// NOTE: Will echo to console the response received from REST Service API
        /// or if an error occured.
        /// </summary>
        /// <param name="message">The raw JSON object that was received from PubNub
library</param>
        /// <returns></returns>
        static async Task ForwardMessageToHTTPService(object message)
        {
            try
            {
                HttpResponseMessage response = await
client.PostAsJsonAsync(ConfigurationManager.AppSettings.Get("REST_API_RequestUri"),
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>";
?>
```