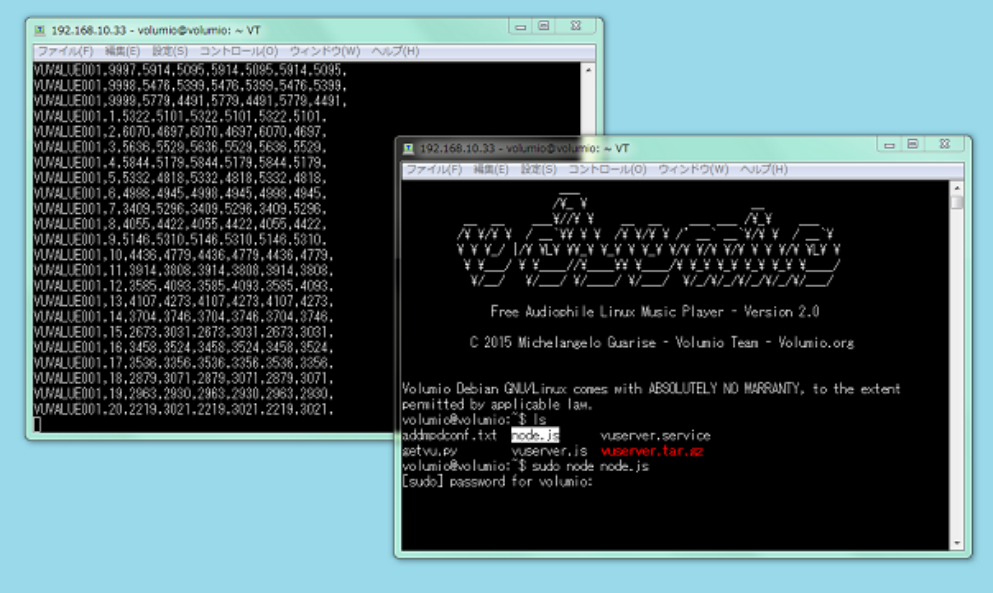
資料：VUServer【追補編】

Pal8000.83001@gmail.com



1. はじめに

本書は、「VUServer 取扱説明書.pdf」の追補情報を記載しています。
対象者は、サーバー側の解析、改変、応用を試したい方です。
今の仕組みで改良したり、サーバー側ソフトを総入れ替えたり、ということをするのに必要と思われる情報になります。
導入したが動かない、という場合のトラブルシュートにもご利用になれます。

内容は、2019/12/5 時点の仕様を基に記載しています。
サーバー：1.1.0 版 20191205
クライアント：3.0.1.0 版
本書の内容は、今後、告知なく改変する可能性があります。

2. 手動起動と動作確認

2-1. モジュール毎のマニュアル起動

Volumio2 側は自動起動のため、通常は特にすることはありませんが、
トラブルシュートやプログラムの 解析、変更したい方は、以下をご参考にしてください。

2-1-1. 自動起動の停止

Raspberry pi 側、一旦自動起動しているのを止めます。

```
#sudo systemctl stop vuserver.service
```

2-1-2. 個別モジュールの起動

TeraTerm 等のターミナルを 3 つ起動し、個別にスクリプトを起動します。
node.js が参照するファイルが必要なので、getvu.py を最初に起動してください。
ブート後に一旦起動した以降であれば順序は問いません。
各スクリプトは Ctrl-C、またはターミナルを終了させることで停止することができます。

2-1-2-1.getvu.py

Python で記述された、リアルタイム音量データを取得するスクリプトです。
起動

```
#sudo sudo python getvu.py
```

これで、Volumio2 を再生すると、getvu.py 側でサンプルした値がコンソールにスクロール表示されます。
自動起動時は、コンソール出力は抑制されます。

2-1-2-2.node.js

node.js で記述された、HTTP サーバー機能のスクリプトです。
起動

```
#sudo sudo node node.js
```

プロンプトが返ってくるだけですが、HTTP サーバーが立ち上がります。

2-1-2-3.syncvu.py

Python で記述された、再生音量と音量データ取得を同期させるスクリプトです。
起動

```
#sudo sudo python syncvu.py
```

内部処理は、音量データ取得先の FIFO を定期的に一瞬 Disable にしているだけです。
これを行わないと、長時間の連続再生で針の振れが先行してしまう様になります。

2-1-3. ブラウザーからの動作確認

Volumio2 で音楽再生を開始してから、Web.ブラウザーからアクセスします。
通常、Volumio2 をアクセスする URL のアドレス部に:8251 を記述して下さい。

例： http://volumio.local:8251

"VUVALUE001"で始まるカンマ区切りのデータが 1 行だけ表示されれば成功です。
読み込み直すと、音量変化に伴い、データの値が変化します。

3. HTTP データの書式

HTTP サーバー側を Raspberry pi で説明してきましたが、リアルタイムの音量情報を以下の書式で返せれば、クライアントアプリはデータ取得して針を振ることができます。

サーバー送信データ書式

```
<html><body>¥n
VALUE001,フレーム番号,LchVU 値,RchVU 値,LchRMS 値,RchRMS 値,LchPeak 値,RchPeak 値,¥n
</body></html>¥n
```

※¥n は改行コードです。

body 部各項目の意味

項目	摘要	クライアント側処理
"VALUE001"	シグニチャーで固定文字列	この文字列がないと問い合わせ先設定エラーになります。
フレーム番号	範囲 1～10000 で、音量をサンプリングするたび 1 カウント増やし、10000 に達したら 1 に戻ります。 Volumio2 の再生が止まると、フレーム番号が変化しなくなります。 HTTP サーバーは最新のフレーム番号を改変せずに中継します。	クライアント側は 3 回続けて同一番号を受信すると再生停止と判定し、各音量データの値によらず音量を 0 とみなします。
各音量データ	カンマ区切り、リニア値で以下の順で入ります。 範囲(分解能)は 0～10000 です。 LchVU 値、RchVU 値、 LchRMS 値、RchRMS 値, LchPeak 値 RchPeak 値, 行末にもカンマが必要です。	VU,RMS,Peak の各 LR チャネルの瞬間音量として処理します。 リニア→dB 値変換はクライアントで行います。

なお、Windows のクライアント側は、HTTP の GET により取得します。
１回の通信では最大 128byte 分を１度取り込むのみで、続き電文の有無を判定しません。
通信設定は、KeepAlive 指定あり、プロキシーなしになっています。

4.ソースコード解説

4-1.getvu,py スクリプトのソース

初版のソースコード

```
import os
import sys
import audioop
import time
import errno

with open('/dev/shm/vulevel.txt', 'w') as f:
      f.write('VUVALUE001,10000,10000,10000,10000,10000,100000,10000,')

time.sleep(3)

# /etc/mpd.conf

while 1:
      fifo = os.open('/tmp/vulevel.fifo', os.O_RDONLY)
      cn = 0
      for cn in range(1, 10000):
            time.sleep(0.02)

            try:
                  rawStream = os.read(fifo, 2048)
            except OSError as err:
                  if err.errno == errno.EAGAIN or err.errno == errno.EWOULDBLOCK:
                        print("exception occured.")
                        rawStream = None
                  else:
                        print("exception occured.")
                        raise

            if rawStream:
                        leftChannel = audioop.tomono(rawStream, 2, 1, 0)
                        rightChannel = audioop.tomono(rawStream, 2, 0, 1)
                        leftPeak = (audioop.max(leftChannel, 2) * 10000) / 32767
                        rightPeak = (audioop.max(rightChannel, 2) * 10000) / 32767
                        #leftRMS = (audioop.rms(leftChannel, 2) * 10000) / 32767
                        #rightRMS = (audioop.rms(rightChannel, 2) * 10000) / 32767
                        #leftAVG = (audioop.avg(leftChannel, 2) * 10000) / 32767
                        #rightAVG = (audioop.avg(rightChannel, 2) * 10000) / 32767
                        s = 'VUVALUE001,{},{},{},{},{},{},{},                    '.format(str(cn), str(leftPeak), str(rightPeak),
str(leftPeak), str(rightPeak), str(leftPeak), str(rightPeak))
                        #leftDB = 20 * math.log10(leftPeak) -74
                        #rightDB = 20 * math.log10(rightPeak) -74
                        if sys.stdout.isatty():
                                print(s)
                        with open('/dev/shm/vulevel.txt', 'rw+b') as f:
                              #with contextlib.closing(mmap.mmap(f.fileno(), 0)) as m:
                                    #m.seek(0)
                                    f.write(s)

      os.close(fifo)
```

参考 *MPD, FIFO, Python, Audioop, Arduino, and Voltmeter: "Faking" a VU Meter*

当初、stack overflow の記事のままで動作したので、ほとんど手を加えていません。

MPD の仕様については、ネット検索で解る情報が少なく手探りをしています。
ここでは初版のスクリプトを元に、不明点、不具合に見えそうな所を以下に解説します。
個々の解説は、ソースのどこを指しているか明示していません。

*4-1-1. * 10000 / 32767*

MPD から取得できる音量を、0～32,767 のリニア値と想定しています。
クライアントの仕様で、サンプルした値を 0～10,000 になる様にスケーリングしています。
経験上、100%音量を 32,767 と想定していますが、仕様上確認が取れていません。

4-1-2. audioop.avg()、audioop.rms() を使っていない

いずれも、Python のライブラリ関数です。
audioop.avg()は平均値を算出してくれますが、絶対値の平均値ではないようで、
マイナス値を含むレンジが狭い値が返ってきます。
audioop.rms()は RMS 値を返しますが、クライアント側で計算するので、利用しません。
今回は audioop.max()の値のみ、VU,RMS,Peak 共通でクライアントに返します。
クライアント側では、VU の移動平均計算、RMS 計算、Peak の針位置制御をします。
リニア→dB 計算もクライアント側で行っています。
クライアント側は 3 種類の値を個別に取得する様にしてあるので、曲の再生位置
インジケータなどに転用することもできます。

4-1-3. メモリマップトファイルを使っていない

アクセスの衝突障害を懸念して、当初は利用する構想でしたが、HTTP サーバー側を
node.js で書き換える際、書き方が判らず、普通のファイルでやり取りするのに留めました。
特に障害は起きておらず、処理負荷も考慮して利用しないことにしました。

4-1-4. HTML ボディの終端にゴミ

数値データ桁数が可変のため、行末にスペースを入れ過去データを上書きしています。
行末に不要なブランクが入りますが、サーバー、クライアント共に、処理を少しでも
減らし負荷軽減することを優先しました。

4-1-5. 3 秒タイマ

起動時に MPD が稼動し、FIFO ファイルの利用可能時期を待つ意図で入っています。
Systemd の設定で、MPD 起動待ちをしているのですが、起動直後に FIFO を開けると
ファイルオープンエラーになります。
3 秒は要らないはずですが、確実な時間を設定しました。

4-1-6. 0.02 秒タイマ

FIFO バッファ取得間隔を決める目的で、タイマを入れてあります。
スクリプトでは 20mS に設定されています。
Windows 側のタイマ処理間隔は最短で 16mS 程度ですので、サーバー側の更新間隔を極端に短くしても意味がありません。20mS 前後が順当と思います。
フレーム番号はここのループ内で採番されます。
タイマ値を変更することで、処理負荷と精度のバランスをある程度調整することが可能です。

4-1-8. Windows 側サンプリング処理

前項の 0.02 秒タイマに関連し、Windows 側の処理を記載します。
LAN 回線を利用するため、回線品質に幅があることを配慮し、Windows 側に電文に乗ってくるフレーム番号により、通信間隔の自動調整を行う機能を装備しています。
具体的には、100 回正常に HTML 受信する度に、1mS ずつ次の通信開始までの間隔を短くします。初期値は 20mS、最低で 1mS の指定になります。
ただし、OS の制限上 16mS 以下の間隔にはならない様です。
逆に、2 回続けて同じフレーム番号を受信すると、通信間隔を 1mS ずつ長くします。
同一番号 3 回目以降は、サーバー側の再生停止とみなし、通信間隔の更新は行いません。
停止判定中の音量は、電文上の音量データに関わらす、0 とみなします。
LAN 環境によるので、明確には言えませんが、現設定で、開発環境では通常 30mS 間隔での通信が成立しています。

4-1-9. FIFO 受信バッファサイズ

2048 としてありますが、MPD の仕様がわかっていないので妥当性は不明です。
20mS 間に MPD が利用するデータ量が判れば、その値を入れれば、ピーク値を取り逃がす恐れと、長い時間ピーク値が続いたという誤測定が少なくなりそうです。
Volumio のディストリビューションでは、入力側が 8192 に設定されているので、最大でも 8192 までと思われますが、MPD 側の設定妥当性も踏まえ、調整の余地はあります。

4-2.node.js (HTTP サーバー)

4-2-1. HTTP サーバーソースコード

node.js のソースは以下の通りです。

```
var http = require('http');
var server = http.createServer(function(req, res) {
   var fs = require('fs');
   fs.readFile('/dev/shm/vulevel.txt', 'utf8', function (err, text) {
      res.write("<html><body>¥n");
      res.write(text);
      res.write("</body></html>¥n");
      res.end();
   })
}).listen(8251);
```

共有ファイルから読み込んだデータを Body 部にし、HTML ヘッダで挟んだだけの処理です。

4-2-2.通信ポートアドレスの変更方法

well-known ポートや登録済ポートを避け、8251 に設定してありますが、変更するには
最終行の listen(8251)の括弧内を任意の値に変更してください。
クライアント側では well-known ポートの範囲を設定できない様にブロックしています。

4-2-3.node.js 採用の経緯

余談ですが、今回要求されるリアルタイム性で、Python が Web サーバーとして実用的に
使えないと判り、次案として、Volumio2 のプラグインに仕立て上げられないか、という
事を考えました。
Volumio 自体が Web.サーバー機能を利用しているので、共有して使えないかと思ったのです。
HELP から調べてみると、node.js で実装している様なので、移植してみました。
結果、実用上問題ない速度で動作するようになりました。

Systemd により自動実行することで、実用的に使えるようになったので、
Volumio2 のプラグイン化に関しては、現時点で踏み込んだ調査を継続していません。

4-3.syncvu.py (音声と針の同期)

長時間の再生で、針の振れが音とずれてくるため、それに対応するスクリプトです。

4-2-1. ソースコード

syncvu.py のソースは/home/volumio にあります。
本書の 1 ページに納まらないので、ここでは割愛します。

ここの処理で MPDClient を使うために、python-mod ライブラリをインストールする
必要があります。
互換性のある新しい派生ライブラリがありますが、処理内容が古いライブラリでも
間に合うので、Volumio との整合性を考慮し、あえて枯れているであろう
古いライブラリを選択しました。

4-2-1. 例外処理

例外処理がいい加減ですが、MPDError 以外の例外はないと踏んで、連続稼動させたところ
インスタンスがいなくなっていた事例がありました。
その後、再現しないのですが、他の要因で例外が発生しても復帰してしまう様に改変して
あります。

4-2-2. 処理内容

処理は、FIFO チャネル番号の割り出しと、定期的に一瞬だけ出力停止をするのみです。
レベル取得用の FIFO チャネルは、/etc/mpd.conf に audio_output を追記することで
追加しましたが、名称でつけた“vulevel”をキーにチャネル番号を特定しています。
そのため、他の用途で audio_output を追加していても問題ありません。
続くループ処理内で FIFO チャネルの出力制御を繰り返し行っています。
コンソールで以下の処理を繰り返しているのと同等です。

```
volumio@volumio2-pi2b:/$ mpc disable 3
Output 1 (alsa) is enabled
Output 2 (multiroom) is disabled
Output 3 (vulevel) is disabled
volumio@volumio2-pi2b:/$ mpc enable 3
Output 1 (alsa) is enabled
Output 2 (multiroom) is disabled
Output 3 (vulevel) is enabled
```

出力先は、マニュアル動作だと 1ch から数えますが、今回のスクリプトでは 0ch から数えます。

4-2-3. 実行間隔

実行間隔設定ですが、Disable が 10mS, Enable が 40 秒を繰り返す様にしています。

当初 120 秒を想定して構築しましたが、通信タイムアウトが発生しやすく、40 秒にしました。
タイムアウト時はリソース解放に時間がかかり、データ計測側のリアルタイム性が損なわれるので、開発環境でのカット＆トライで、タイムアウトが発生しない短めの時間設定になっています。
エラーリカバー自体はする様にしてあります。

5.動作不良と思ったら

5-1.針が動かない

Volumio2 の設定を変更すると、/etc/mpd.conf が書き戻される場合があります。
再生デバイス周りを変更したら確実に書き戻されますので、取扱説明書に従い
再設定してください。

5-2.針の動きが鈍い

回線状態により、データ取得間隔が想定以上に長くなってしまっている可能性があります。
クライアント(VUMeter for Windows)を停止、再起動してみてください。

5-3.クライアントにポップアップエラー

サーバー接続設定中であれば、正しく繋がるまでは繰り返しポップアップが表示されます。
それまで動作していたのであれば、ポップアップを閉じて様子を見てください。
Volumio2 側が落ちている事が主たる原因ですが、回線状態により、一時的にサーバーとの
接続が絶たれたと判定されることがあります。

版数履歴

2019/2/15　初版

2019/12/5　改訂