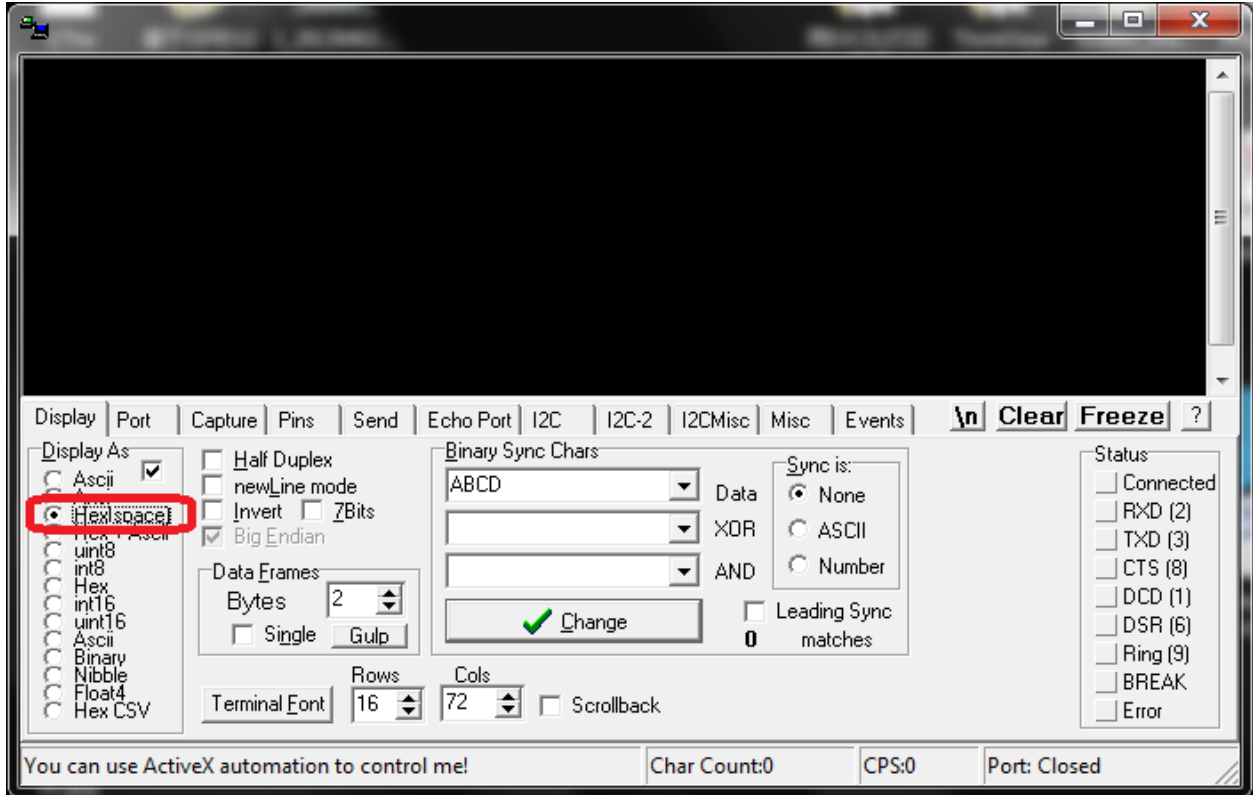

TGAM数据流格式说明

April 3, 2014

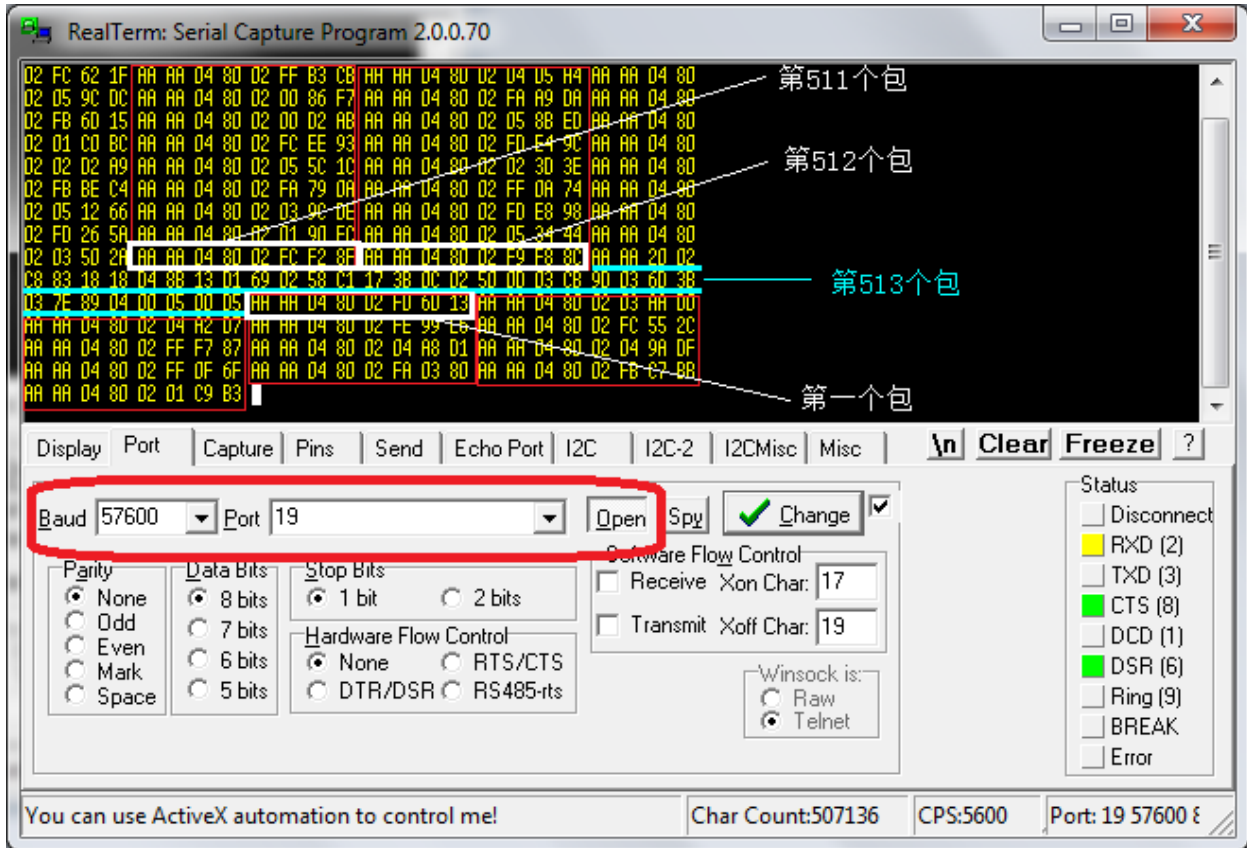
1. 建议用RealTerm从串口抓数据，下载地址：

http://realterm.sourceforge.net/index.html#downloads_Download

2. 打开RealTerm，指定显示方式：在Displa标签页，选择Hex+space。下图中红圈标记的地方：



3.指定波特率，端口，点击Open按钮。如果连接成功，你应该会看到类似下图的数据。



4.说明：

TGAM大约每秒钟发送513个包，注意是“大约每秒钟”，意思就是发送包的个数是不会变的，只是发送513个包所花费的时间是一秒左右。

发送的包有小包和大包两种：小包的格式是AA AA 04 80 02 xxHigh xxLow xxChecksum前面的AA AA 04 80 02是不变的，后三个字节是一只变化的，xxHigh和xxLow组成了原始数据rawdata，xxChecksum就是校验和。所以一个小包里面只包含了一个对开发者来说有用的数据，那就是rawdata，可以说一个小包就是一个原始数据，大约每秒钟会有512个原始数据。

那怎么从小包中解析出原始数据呢？ $rawdata = (xxHigh \ll 8) | xxLow;$

$if(rawdata > 32768)\{ rawdata -=65536;\}$

现在原始数据就这么算出来了，但是在算原始数据之前，我们先应该检查校验和。校验

和怎么算呢？ $sum = ((0x80 + 0x02 + xxHigh + xxLow) \wedge 0xFFFFFFFF) \& 0xFF$

什么意思呢？就是把04后面的四个字节加起来，取反，再取低八位。

如果算出来的sum和xxChecksum是相等的，那说明这个包是正确的，然后再去计算rawdata，否则直接忽略这个包。丢包率在10%以下是不会对最后结果造成影响的。

现在，原始数据出来了，那我们怎么拿信号强度Signal,专注度Attention,放松度Meditation,和8个EEG Power的值呢？就在第513个这个大包里面，这个大包的格式是相当固定的，我们就拿上图中的数据来一个字节一个字节地说明他们代表的含义：

AA 同步

AA 同步

20 是十进制的32，即有32个字节的payload，除掉20本身+两个AA同步+最后校验和

02 代表信号值Signal

C8 信号的值

83 代表EEG Power开始了

18 是十进制的24，说明EEG Power是由24个字节组成的，以下每三个字节为一组

18 Delta 1/3

D4 Delta 2/3

8B Delta 3/3

13 Theta 1/3

D1 Theta 2/3

69 Theta 3/3

02 LowAlpha 1/3

58 LowAlpha 2/3

C1 LowAlpha 3/3

17 HighAlpha 1/3

3B HighAlpha 2/3

DC HighAlpha 3/3

02 LowBeta 1/3

50 LowBeta 2/3

00 LowBeta 3/3

03 HighBeta 1/3

CB HighBeta 2/3

9D HighBeta 3/3

03 LowGamma 1/3

6D LowGamma 2/3

3B LowGamma 3/3

03 MiddleGamma 1/3

7E MiddleGamma 2/3

89 MiddleGamma 3/3

04 代表专注度Attention

00 Attention的值(0到100之间)

05 代表放松度Meditation

00 Meditation的值(0到100之间)

D5 校验和

以下C#代码说明怎么解析数据。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ReadParseTGAM
{
    public class Parser
    {

        public const int PARSER_CODE_POOR_SIGNAL = 2;
        public const int PARSER_CODE_HEARTRATE = 3;
        public const int PARSER_CODE_CONFIGURATION = 4;
        public const int PARSER_CODE_RAW = 128;
        public const int PARSER_CODE_DEBUG_ONE = 132;
        public const int PARSER_CODE_DEBUG_TWO = 133;
        public const int PARSER_CODE_EEG_POWER = 131;

        public const int PST_PACKET_CHECKSUM_FAILED = -2;
        public const int PST_NOT_YET_COMPLETE_PACKET = 0;
        public const int PST_PACKET_PARSED_SUCCESS = 1;
        public const int MESSAGE_READ_RAW_DATA_PACKET = 17;
        public const int MESSAGE_READ_DIGEST_DATA_PACKET = 18;

        private const int RAW_DATA_BYTE_LENGTH = 2;
        private const int EEG_DEBUG_ONE_BYTE_LENGTH = 5;
        private const int EEG_DEBUG_TWO_BYTE_LENGTH = 3;
        private const int PARSER_SYNC_BYTE = 170;
        private const int PARSER_EXCODE_BYTE = 85;
        private const int MULTI_BYTE_CODE_THRESHOLD = 127;
        private const int PARSER_STATE_SYNC = 1;
        private const int PARSER_STATE_SYNC_CHECK = 2;
        private const int PARSER_STATE_PAYLOAD_LENGTH = 3;
        private const int PARSER_STATE_PAYLOAD = 4;
        private const int PARSER_STATE_CHKSUM = 5;

        private int parserStatus;
        private int payloadLength;
        private int payloadBytesReceived;
        private int payloadSum;
```

```

private int checksum;
private byte[] payload = new byte[256];

public Parser()
{
    this.parserStatus = PARSER_STATE_SYNC;
}

public int parseByte(byte buffer)
{
    int returnValue = 0;
    switch (this.parserStatus)
    {
        case 1:
            if ((buffer & 0xFF) != PARSER_SYNC_BYTE) break; this.parserStatus =
PARSER_STATE_SYNC_CHECK;
            break;
        case 2:
            if ((buffer & 0xFF) == PARSER_SYNC_BYTE)
                this.parserStatus = PARSER_STATE_PAYLOAD_LENGTH;
            else
            {
                this.parserStatus = PARSER_STATE_SYNC;
            }
            break;
        case 3:
            this.payloadLength = (buffer & 0xFF);
            this.payloadBytesReceived = 0;
            this.payloadSum = 0;
            this.parserStatus = PARSER_STATE_PAYLOAD;
            break;
        case 4:
            this.payload[(this.payloadBytesReceived++)] = buffer;
            this.payloadSum += (buffer & 0xFF);
            if (this.payloadBytesReceived < this.payloadLength) break; this.parserStatus =
PARSER_STATE_CHKSUM;
            break;
        case 5:
            this.checksum = (buffer & 0xFF);
            this.parserStatus = PARSER_STATE_SYNC;
            if (this.checksum != ((this.payloadSum ^ 0xFFFFFFFF) & 0xFF))
            {

```

```

        returnValue = -2;
        Console.WriteLine("Checksum ERROR!!");
    }
    else
    {
        returnValue = 1;
        parsePacketPayload();
    }
    break;
}
return returnValue;
}

private void parsePacketPayload()
{
    int i = 0;
    int extendedCodeLevel = 0;
    int code = 0;
    int valueBytesLength = 0;

    int signal = 0; int config = 0; int heartrate = 0;
    int rawWaveData = 0;
    while (i < this.payloadLength)
    {
        extendedCodeLevel++;

        while (this.payload[i] == PARSER_EXCODE_BYTE)
        {
            i++;
        }

        code = this.payload[(i++)] & 0xFF;

        if (code > MULTI_BYTE_CODE_THRESHOLD)
        {
            valueBytesLength = this.payload[(i++)] & 0xFF;
        }
        else
        {
            valueBytesLength = 1;
        }

        if (code == PARSER_CODE_RAW)

```

```

{
  if ((valueBytesLength == RAW_DATA_BYTE_LENGTH))
  {
    byte highOrderByte = this.payload[i];
    byte lowOrderByte = this.payload[(i + 1)];

    rawWaveData = getRawWaveValue(highOrderByte, lowOrderByte);

    if (rawWaveData > 32768) rawWaveData -= 65536;

    Console.WriteLine("Raw:"+rawWaveData);

  }
  i += valueBytesLength;
}
else
{
  switch (code)
  {
    case PARSE_CODE_POOR_SIGNAL:
      signal = this.payload[i] & 0xFF;
      i += valueBytesLength;
      Console.Write("PQ:" + signal);
      break;
    case PARSE_CODE_EEG_POWER:
      i += valueBytesLength;
      break;
    case PARSE_CODE_CONFIGURATION:
//Signal 等于以下值，代表耳机没有戴好
      if ( signal == 29 || signal == 54 || signal == 55 || signal == 56 || signal == 80 ||
signal == 81 || signal == 82 || signal == 107 || signal == 200)
      {
        config = this.payload[i] & 0xFF;

        Console.Write("--NoShouldAtt:" + config);

        Console.WriteLine("");

        i += valueBytesLength;

        break;
      }
    else

```



```

        {

            config = this.payload[i] & 0xFF;

            Console.Write("--Att:" + config);
            Console.WriteLine("");

        }

        i += valueBytesLength;
        break;
    case PARSER_CODE_HEARTRATE:
        heartrate = this.payload[i] & 0xFF;
        i += valueBytesLength;

        break;
    case PARSER_CODE_DEBUG_ONE:
        if (valueBytesLength == EEG_DEBUG_ONE_BYTE_LENGTH)
        {
            i += valueBytesLength;
        }
        break;
    case PARSER_CODE_DEBUG_TWO:
        if (valueBytesLength == EEG_DEBUG_TWO_BYTE_LENGTH)
        {
            i += valueBytesLength;
        }
        break;
    }
}
}
this.parserStatus = PARSER_STATE_SYNC;
}

```

```

private int getRawWaveValue(byte highOrderByte, byte lowOrderByte)

```

```

{
    /* Sign-extend the signed high byte to the width of a signed int */
    int hi = (int)highOrderByte;

    /* Extend low to the width of an int, but keep exact bits instead of sign-extending */
    int lo = ((int)lowOrderByte) & 0xFF;

    /* Calculate raw value by appending the exact low bits to the sign-extended high bits */

```

```
int value = (hi << 8) | lo;
```

```
return (value);
```

```
}
```

```
}
```

```
}
```