# ProtocolParser 类使用说明

## 功能描述

- `ProtocolParser`类实现了对满足`FrequencyAnalyseV3.0`协议规范的数据流的**连接**、**解析**与**校验**;
- 封装了报文发送函数，可以方便地使用`ProtocolParser`类接口函数，**发送**满足协议规范的报文。

## 函数接口

### 类构造函数

- ```
  public ProtocolParser();
  ```

  类构造函数，程序开始时创建类的实例即可:

  ```
  ProtocolParser parser = new ProtocolParser();
  ```

  一个数据流可以同时feed多个解析类，但一个解析类只用于一个数据流的解析。

### 数据接收相关函数

- ```
  public byte ParsingMessage(byte[] msg, int len)
  ```

  数据输入流解析函数。该函数获取数据流输入，每次的输入数据可以是任意长度的byte，可以是截断的报文，只要输入的byte是**保序**的，函数内部会对输入数据进行缓存和连接；当确认接收到一个完整报文时，函数会自动解析报文，校验报文正确性，并调用对应报文的接收处理函数，见下表：

| 报头 | 报文内容 | 调用函数名 |
| --- | --- | --- |
| `0x51`、`0x52`、`0x53` | 3个频率通道的频率数据 | `void frequency_callback(int channel, WavePara freq)` |
| `0x59` | 电池电量状态 | `void battery_callback(BatteryStatus Batt)` |
| `0x5F` | 3个频率通道的使能指令 | `void channel_enable_callback(boolean[] channelEnable)` |

  使用该函数时，只需将输入数据流以`byte[]`形式传递给函数，为应对并非`byte[]`中所有`byte`均为有效数据的情况，需要在函数第二个参数中指定输入的`byte[]`包含的有效数据长度，用法如下:

```
BufferedInputStream binInput = new BufferedInputStream(new
FileInputStream("Data.dat"));
ProtocolParser parser = new ProtocolParser();
int byteRead;
byte[] msg;

while ((byteRead = binInput.available()) > 0) {
    msg = binInput.readNBytes(byteRead);
    parser.ParsingMessage(msg, byteRead);
}
```

- ```
  private void frequency_callback(int channel, WavePara freq)
  ```

接收到一帧频率报文时回调的函数，使用时在该函数内部实现收到该报文时需要做出的响应。传入的数据类`WavePara`封装了通道`channel`（1~3）的频率信息。`WavePara`类定义如下：

```
class WavePara {
    public int t;
    public float freq;
    public float mag;
    public float freq_deriv;

    public WavePara() {
        t = 0;
        freq = 0;
        mag = 0;
        freq_deriv = 0;
    }
};
```

- ```
  private void battery_callback(BatteryStatus Batt)
  ```

接收到一帧电池电量状态报文时回调的函数，使用时在该函数内部实现收到电量报文时需要做出的响应。传入的数据类`BatteryStatus`封装了所有电池的电量状态信息。`BatteryStatus`类定义如下：

```
class BatteryStatus {
    public int t;
    public short voltage;
    public short current;
    public byte capacity;

    public BatteryStatus() {
        t = 0;
        voltage = 0;
        current = 0;
        capacity = 0;
    }
};
```

- ```
  private void channel_enable_callback(boolean[] channelEnable)
  ```
  ```

接收到一帧电池电量状态报文时回调的函数。该函数供嵌入式系统调用，响应手机端上位机发来的通道使能指令，Android端可不关注，保持该函数为空。

## 数据发送相关函数

- ```
  public void send_channel_enable_info(boolean[] channelEnable)
  ```

  以协议格式发送3个频率通道的使能指令报文。该函数由Android端上位机调用，`channelEnable[i] == false`时，接收到报文的下位机将禁能频率通道`i`，不再发送该通道的频率数据；`channelEnable[i] == true`时可再次使能。

- ```
  public void send_frequency_info(int channel, WavePara wave)
  ```

  以协议格式发送3个频率通道的频率数据报文。该函数由嵌入式端调用，Android端不调用。

- ```
  public void send_battery_info(BatteryStatus battery)
  ```

  以协议格式发送电池电量状态报文。该函数由嵌入式端调用，Android端不调用。

- ```
  private void putc_callback(byte[] msg)
  ```

  输出数据流的实现函数。前文提及的3个报文发送函数都依赖于本函数实现字节流的发送。需要由函数使用者实现发送数据流的功能。输出数据存放于`byte[] msg`数组中，长度为`msg.length`，待发送的数据低地址先发送，高地址后发送，`msg[0]`->`msg[msg.length]`。

# 用法说明

对于Android端上位机，只**需实现**以下函数：

```
private void frequency_callback(int channel, WavePara freq);
private void battery_callback(BatteryStatus Batt);
private void putc_callback(byte[] msg);
```

Android端与嵌入式端的数据流如下图：