ThinkGear Socket Protocol

June 28, 2009

NeuroSky®
Brain-Computer Interface Technologies

The NeuroSky product families consist of hardware and software components for simple integration of this biosensor technology into consumer and industrial end-applications. All products are designed and manufactured to meet exacting consumer specifications for quality, pricing, and feature sets. NeuroSky sets itself apart by providing building-block component solutions that offer friendly synergies with related and complementary technological solutions.

Reproduction in any manner whatsoever without the written permission of NeuroSky Inc. is strictly forbidden. Trademarks used in this text: eSense™, ThinkGear™, MDT™, NeuroBoy™and NeuroSky™are trademarks of NeuroSky Inc.

NO WARRANTIES: THE DOCUMENTATION PROVIDED IS "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, INCLUDING PATENTS, COPYRIGHTS OR OTHERWISE, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT SHALL NEUROSKY OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, COST OF REPLACEMENT GOODS OR LOSS OF OR DAMAGE TO INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE DOCUMENTATION PROVIDED, EVEN IF NEUROSKY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. , SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES.

Contents

June 28, 2009 | © 2009 NeuroSky, Inc. All Rights Reserved.

Introduction

This document defines the protocol for communicating with devices or software that implement the ThinkGear Socket Protocol (TGSP), such as the ThinkGear Connector (TGC). The TGSP was designed to allow languages and/or frameworks without a standard serial port API (e.g. Flash and most scripting languages) to integrate brainwave-sensing functionality.

TGSP streams data read from ThinkGear technology, such as the MindSet headset, to an open socket via TCP. Any language and/or framework that has a socket library will thus be able to communicate with the headset through this open socket.

Important: This document is a **draft specification** and is *extremely likely* to change prior to the final release of the document.

Conventions

Before delving into the packet specification, it is important to establish a few formatting conventions used in this document to facilitate explanation.

`[TYPE]`

Surrounding a `TYPE` with square brackets means that the `TYPE` is exactly a single byte long.

`[TYPE...]`

A `TYPE` trailed by an ellipsis indicates that the `TYPE` consists of one or more bytes.

`([TYPE])`

`TYPE` byte(s) surrounded by parentheses indicates that this byte may or may not be present, depending on the context.

June 28, 2009 | © 2009 NeuroSky, Inc. All Rights Reserved.

Packet Syntax

TGSP uses a packet structure similar to that of the ThinkGear Communications Protocol (TGCP) — in fact, the PAYLOAD portion of the structure is largely identical. The primary difference is that TGSP does not contain a checksum of the packet, which is used in the TGCP for error-checking.

Structure

```
[HEADER...] [PAYLOAD...]
```

Every packet sent by a TGSP server will contain a fixed-length header and a variable-length payload.

Header

```
[SYNC] [SYNC]
```

The header for each packet consists of two SYNC bytes, having a value of 0xAA. It is useful for determining the start of a packet, in the case where the implementing application loses its place in the buffer and needs to realign itself. In general, one should scan the buffer for these two SYNC bytes before starting to parse the payload.

Payload

```
[CODE] ([LENGTH]) [DATA...]
```

Every packet payload contains any number of the above tuple, where the exact number of tuples per packet is dictated by how the ThinkGear has been configured. The CODE and the optional LENGTH are always a single unsigned byte, whereas DATA can be a variable number of bytes long.The CODE portion of the packet dictates whether there is a LENGTH portion, and also how the DATA portion of the packet should be parsed. The following table lists the available CODEs, their corresponding LENGTH and DATA, and a short description of their function.

June 28, 2009 | © 2009 NeuroSky, Inc. All Rights Reserved.

CODE	DATA content	LENGTH	Description
`0x02`	One (1) unsigned byte	Not used	**Poor signal** level
`0x04`	One (1) unsigned byte	Not used	**eSense Attention** value
`0x05`	One (1) unsigned byte	Not used	**eSense Meditation** value
`0x81`	Eight (8) big-endian floats	`0x20`	**EEG powers**, corresponding to delta, theta, low beta, high beta, low alpha, high alpha, low gamma, and high gamma, respectively

Note that there is **no guarantee** of payload order; EEG powers may come before the eSense values, for example.

Example Packet

Let us examine a standard packet sent by a TGSP server to an application. The raw output of the byte stream looks like the following (interpreted as hexadecimal numbers):

```
AA AA 02 1A 04 27 05 62 81 20 38 F1 50 C1 35 BD C0 55 39 0D A7
A7 38 8C 51 78 37 78 35 C6 35 3A CC CF 35 0D 61 CD 37 6C 1B 71
```

Here is the interpretation of each byte:

June 28, 2009 | © 2009 NeuroSky, Inc. All Rights Reserved.

Index	Byte	Description
0	`0xAA`	`SYNC`
1	`0xAA`	`SYNC`
2	`0x02`	**Poor signal** `CODE`
3	`0x1A`	**Poor signal** level of 26
4	`0x04`	**eSense attention** `CODE`
5	`0x27`	**eSense attention** value of 39
6	`0x05`	**eSense meditation** `CODE`
7	`0x62`	**eSense meditation** value of 98
8	`0x81`	**EEG powers** `CODE`
9	`0x20`	**EEG powers** `LENGTH`
10-13	`0x38F150C1`	**EEG delta** value of `1.15e-4`
14-17	`0x35BDC055`	**EEG theta** value of `1.41e-6`
18-21	`0x390DA7A7`	**EEG low alpha** value of `1.35e-4`
22-25	`0x388C5178`	**EEG high alpha** value of `6.69e-5`
26-29	`0x377835C6`	**EEG low beta** value of `1.47e-5`
30-33	`0x353ACCCF`	**EEG high beta** value of `6.95e-7`
34-37	`0x350D61CD`	**EEG low gamma** value of `5.26e-7`
38-41	`0x376C1B71`	**EEG high gamma** value of `1.40e-5`

June 28, 2009 | © 2009 NeuroSky, Inc. All Rights Reserved.

Parsing

By default, a TGSP server will transmit a packet of data every second (1Hz). Any parser implementations can thus keep the main parsing loop in a busy-wait state for most of the time.

Pseudocode

The pseudocode for the parser is as follows:

```
while the socket connection is open and there is data in the buffer
   scan the buffer until the sync bytes are seen

   while there is still data in the socket buffer
      read a code byte

      if the code byte refers to "attention"
         read the attention value byte
      else if the code byte refers to "meditation"
         read the meditation value byte
      else if the code byte refers to ...
         ...
      else if the code byte refers to "EEG powers"
         read eight EEG power floating point numbers
      else if the code byte is unrecognized
         continue the loop

   sleep for 1 second
```

The above pseudocode applies to most generic procedural languages. Note that in frameworks that support event handlers (such as ActionScript), the structure of the code can look slightly different.

Example Code

The following examples demonstrate functional implementations of TGSP parsers in a variety of languages.

ActionScript 3.0

ActionScript 3.0 (AS3) is commonly used in application frameworks from Adobe, such as Adobe Flash, Adobe Flex, and Adobe AIR. The following portion of code uses AS3's Socket class, which was a new introduction to ActionScript. As such, Adobe application frameworks that use older versions of ActionScript will not be able to connect to TGSS.

```
/**
 * Class constructor
```


© 2009 NeuroSky, Inc. All Rights Reserved.

```
 */
public function YourClass(){
  // set up a socket connection to TGSS (defaults to localhost:13854)
  thinkGearSocket = new Socket("127.0.0.1", 13854);

  // add socket event listeners
  thinkGearSocket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
}

/**
 * Event handler for when data is received on the socket.
 */
private function dataHandler(e : ProgressEvent){
  // buffer to store the headset data
  var buffer : ByteArray = new ByteArray();

  // read the bytes from the socket into the buffer
  thinkGearSocket.readBytes(buffer, 0, thinkGearSocket.bytesAvailable);

  buffer.position = 1;                    // set buffer position to anticipate SYNC bytes
  buffer.endian = Endian.BIG_ENDIAN;      // set float interpretation to big-endian

  // let's look for the SYNC bytes.
  // the loop consumes the buffer until two consecutive 0xAA bytes are seen
  while(buffer.position < buffer.length &&
        buffer.readUnsignedByte() != 0xAA && buffer[buffer.position - 1] != 0xAA){
    continue;
  }

  // now process the buffer.
  while(buffer.position < buffer.length){
    // read the CODE byte
    var code : uint = buffer.readUnsignedByte();

    switch(code){
      // signal quality - read a single byte
      case 0x02:
        poorSignal = buffer.readUnsignedByte();
        break;
      // attention - read a single byte
      case 0x04:
        attention = buffer.readUnsignedByte();
        break;
      // meditation - read a single byte
      case 0x05:
        meditation = buffer.readUnsignedByte();
        break;
      // EEG powers - read the length, and eight floats (8 * 4 bytes)
      case 0x81:
        var eegPowerLength : Number = buffer.readUnsignedByte();
        delta = buffer.readFloat();
        theta = buffer.readFloat();
        beta1 = buffer.readFloat();
        beta2 = buffer.readFloat();
        alpha1 = buffer.readFloat();
        alpha2 = buffer.readFloat();
        gamma1 = buffer.readFloat();
        gamma2 = buffer.readFloat();
```


June 28, 2009 | © 2009 NeuroSky, Inc. All Rights Reserved.

```
        break;
    }
  }

  // clear the buffer of any contents, in case there are several payload
  // tuples that aren't handled by the parser
  thinkGearSocket.flush();
}
```

 © 2009 NeuroSky, Inc. All Rights Reserved.