

ThinkGear Socket Protocol

June 28, 2010



The NeuroSky product families consist of hardware and software components for simple integration of this bio-sensor 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, NON-INFRINGEMENT 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

Introduction	4
Conventions	4
Overview	5
Authorization	6
Parameters	6
Response	6
Configuration	7
Parameters	7
Response	7
Headset Data Transmission	8
Response	8
Parsing	10
ActionScript 3 (Adobe Flash and Flex)	10
C# (.NET and Mono)	11

Introduction

The ThinkGear Socket Protocol (TGSP) is a [JSON-based](#) protocol for the transmission and receipt of ThinkGear brainwave data between a client and a server. TGSP was designed to allow languages and/or frameworks without a standard serial port API (e.g. Flash and most scripting languages) to easily integrate brainwave-sensing functionality through socket APIs.

This document is a specification for TGSP.

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

Conventions

There will be several nomenclature conventions that will be used throughout this document.

- A **server** is a device or application that implements TGSP, and is responsible, amongst other things, for responding to authorization requests and broadcasting headset data. The ThinkGear Connector is an example of a "server".
- A **client** is a device or application that connects to a server.
- **Headset data** refers to the data returned by a headset containing a ThinkGear module.

JSON nomenclature conventions will also be used throughout this document (primarily the concept of a JSON *object*), so it is best to [scan the language specification](#) to brush up.

Overview

There are several primary stages in the lifetime of a TGSP connection.

1. **Creation of socket connection**
2. **Authorization** (one-time) — Authorization of the client by the server
3. **Configuration of server** (performed any time)
4. **Receipt of headset data** (repeating)
5. **Termination of socket connection**

These primary stages are covered in detail in the following sections.

Authorization

The client must initiate an authorization request and the server must authorize the client before the server will start transmitting any headset data.

Parameters

- `appName`. **Required.** A human-readable name identifying the client application. This can be a maximum of 255 characters.
- `appKey`. **Required.** The key used by the client application to identify itself. This must be 40 hexadecimal characters, ideally generated using an SHA-1 digest. See the Note below.

```
{"appName": "Brainwave Shooters", "appKey": "9f54141b4b4c567c558d3a76cb8d715cbde03096"}
```

Note: The `appKey` is an identifier that is unique to each application, rather than each *instance* of an application. It is used by the server to bypass the authorization process if a user had previously authorized the requesting client. To reduce the chance of overlap with the `appKey` of other applications, the `appKey` should be generated using an SHA-1 digest.

Response

The server will respond to the client after receiving an authorization request from the client. The response will be sent prior to the transmission of any headset data.

- `isAuthorized`. Tells the client whether the server has authorized access to the user's headset data. The value is either `true` or `false`.

```
{"isAuthorized": true}
```

Note: There is **no guarantee** that a response to the authorization request will be transmitted by the server in any amount of time. As such, clients should stay in an idle state until a response is received from the server.

Configuration

A client can send commands to a server to configure such things as transmission formats or the components of data transmitted by the server. These commands can be sent at any time after the authorization process.

Parameters

- `enableRawOutput`. *Optional*. Whether raw sensor output should be included in the transmitted data. The value of this parameter should be either `true` or `false` (default).
- `format`. *Optional*. The format in which headset data should be transmitted to the client. The value of this parameter should be either `"BinaryPacket"` (default) or `"Json"`. When specifying this value, **make note of the capitalization!**

```
{"enableRawOutput": true, "format": "Json"}
```

Response

No explicit response to these packets will be sent by the server — the server will simply start transmitting data in the configured format.

Important: Because it may take some time for the ThinkGear Connector to re-configure itself to transmit JSON packets, several binary packets may be prematurely transmitted to the application. As such, an application should be able to handle the receipt of unexpected binary packets without failing critically.

Headset Data Transmission

Data transmission from the server is done using a streaming model; the client does not issue any explicit requests to the server for brainwave data.

Because there is no mechanism in JSON to handle streaming (i.e. continuously appended) data, TGSP delimits individual JSON objects with carriage return characters (`\r`), so each JSON object will occupy its own line.

Important: Even though JSON is the preferred transmission format, the **binary packet format** (used in earlier versions of TGSP) is the **default** format. Documentation for the binary packet format can be found in the [Binary Socket Packet Format](#) document.

The Binary Socket Packet Format will **eventually be deprecated** in favor of the JSON format, so application developers are encouraged to switch to the JSON format as soon as possible.

Response

- `poorSignalLevel`. A quantifier of the quality of the brainwave signal. This is an integer value that is generally in the range of 0 to 200, with 0 indicating a good signal and 200 indicating an off-head state.
- `eSense`. A container for the eSense™ attributes. These are integer values between 0 and 100, where 0 is perceived as a lack of that attribute and 100 is an excess of that attribute.
 - `attention`. The eSense Attention value.
 - `meditation`. The eSense Meditation value.
- `eegPower`. A container for the EEG powers. These may be either integer or floating-point values.
 - `delta`. The "delta" band of EEG.
 - `theta`. The "theta" band of EEG.
 - `lowAlpha`. The "low alpha" band of EEG.
 - `highAlpha`. The "high alpha" band of EEG.
 - `lowBeta`. The "low beta" band of EEG.
 - `highBeta`. The "high beta" band of EEG.
 - `lowGamma`. The "low gamma" band of EEG.
 - `highGamma`. The "high gamma" band of EEG.

- `rawEeg`. The raw data reading off the forehead sensor. This may be either an integer or a floating-point value. This data is represented in its own JSON object, as in the sample below.
- `blinkStrength`. The strength of a detected blink. This is an integer in the range of 0-255. This data is represented in its own JSON object, as in the sample below.

```
{ "poorSignalLevel": 0, "eSense": { "attention": 38, "meditation": 43 }, "eegPower": { "delta": 1.15e-4, "theta": 1.2e-4 },  
  { "rawEeg": 238 }  
  { "rawEeg": 282 }  
  { "blinkStrength": 100 }  
  { "rawEeg": 239 }
```

Note: With the exception of `rawEeg` and `blinkStrength`, the headset components are transmitted at a rate of 1Hz. `rawEeg`, if enabled, is transmitted at a rate no higher than 512Hz. `blinkStrength` is transmitted whenever a blink is detected by the headset.

Note: The client should **not** expect a specific component of headset data to be present in all (or even any) packets transmitted by the server. The client should thus maintain state between receipts of headset data from the server. Also, the ordering of the parameters in each individual JSON object cannot be guaranteed.

Parsing

Clients will first have to tokenize the stream using the carriage return (`\r`) delimiter, then parse each token individually as a JSON object. This is demonstrated by the following pseudocode:

```
while there is still data in the stream
    read the line
    parse the line as JSON
```

Important: When using the JSON output format and tokenizing a packet stream using a `\r` delimiter, be careful about parsing the last token as a JSON object. The packet stream will end in a `\r` character, meaning that the tokenizer will likely return an empty string as the last token.

Also, your parsing code should be tolerant of incomplete packet strings, in the event that the stream is parsed mid-transfer.

Once a JSON object has been extracted out of the stream, it can be parsed using any of a number of readily-available JSON parsing libraries. An exhaustive list of JSON parsers for various languages can be found at the [JSON website](#), but here are the ones that NeuroSky recommends:

Language	Library
ActionScript 3 (Flash/Flex)	ActionScript 3 corelib
C# (.NET/Mono)	Jayrock

ActionScript 3 (Adobe Flash and Flex)

Once a `Socket` has been created in your code, you'll need to configure the ThinkGear Connector to output JSON (and optionally, raw sensor data). This is done by sending a packet that is formatted to the [Configuration packet specification](#). For example:

```
var configuration : Object = new Object();
configuration["enableRawOutput"] = true;
configuration["format"] = "Json";

socket.writeUTFBytes(JSON.encode(configuration));
```

When reading data from the ThinkGear Connector, you can read data directly into a `String` from the socket stream. For AS3, this code would typically go into the function that was delegated as the event listener:

```
var packetString : String = socket.readUTFBytes(socket.bytesAvailable);
```

Then, the string should be tokenized using the carriage return (`\r`) as the delimiter:

```
var packets : Array = packetString.split(/\r/);
```

You can then iterate over each of the packets, parsing it into JSON:

```
for(var packet : String in packets){
    var data : Object = JSON.decode(packet);

    // note that not all packets will contain a "rawEeg" parameter; the
    // appropriate error checking should be performed.
    trace(data["rawEeg"]);
    trace(data["eSense"]["attention"]);
}
```

C# (.NET and Mono)

Typically, socket data is returned as an array of bytes in a buffer. This should be converted to a `string` prior to parsing it as JSON:

```
byte[] buffer = new byte[8192];
networkStream.Read(buffer, 0, buffer.Length);

string packetString = System.Text.ASCIIEncoding.ASCII.GetString(buffer);
```

Next, the string should be tokenized using a carriage return (`\r`) as the delimiter:

```
string[] packets = String.Split(packetString, new char[] { "\r" });
```

Now that you've split the packet stream into its constituent packets, you can loop over the array and parse each packet individually. The headset data can then be referenced directly:

```
foreach(string packet in packets){
    IDictionary data = (IDictionary)JsonConvert.Import(typeof(IDictionary), packet);

    // note that not all packets will contain a "rawEeg" parameter; the
    // appropriate error checking should be performed.
    Console.WriteLine("Raw data: " + data["rawEeg"]);
}
```

Note: By default, Visual Studio imports the `System.Collections.Generic` package when creating a new class file. During compilation, however, this causes problems with the typecast used above. Simply remove the `import System.Collections.Generic` statement from the file header to fix the compilation error.