1. THINKGEAR ASIC COMMUNICATION DRIVERS AND PROTOCOLS DESCRIPTION

ThinkGear™ is the technology inside every NeuroSky/Brainsense product or partner product that enables a device to interface with the wearers' brainwaves. It includes the sensor that touches the forehead, the contact and reference points located on the ear pad, and the onboard chip that processes all of the data and provides this data to software and applications in digital form. Both the raw brainwaves and the eSense Meters (Attention and Meditation) are calculated on the ThinkGear chip.

Note: The Baud Rate of Brainsense is 9600 whereas of Mindset it is 57600 or 115200.

1.1 ThinkGear Communication Driver (TGCD)

The ThinkGear Connection Driver (TGCD) is a native Windows and OS X library that handles all the "heavy lifting" of interacting with a Brainsense/Mindset, from setting up the connection to interpreting the data stream received from the Brainsense/Mindset. The API exposed by TGCD is extremely simple.

TGCD is distributed as a .DLL (for Windows) and a .bundle (for OS X), making it suitable for applications written in C or C derivatives (i.e. C++, C#, or Objective-C).

ThinkGear.h Header File

This header file defines the ThinkGear Communications Driver (TGCD) API, which is a set of functions for users to connect to and receive data from a ThinkGear data stream.

There are three "tiers" of functions in this API, where functions of a lower "tier" should not be called until an appropriate function in the "tier" above it is called first:

Tier 1: Call anytime
- TG_GetDriverVersion()
- TG_GetNewConnectionId()

Tier 2: Requires TG_GetNewConnectionId()
- TG_SetStreamLog()
- TG_SetDataLog()
- TG_EnableLowPassFilter()
- TG_Connect()
- TG_FreeConnection()

Tier 3: Requires TG_Connect()
- TG_ReadPackets()
- TG_GetValueStatus()
- TG_GetValue()
- TG_SendByte()
- TG_SetBaudrate()
- TG_SetDataFormat()
- TG_Disconnect()

1.1.1 ThinkGear Header file Macro Definition Documentation

#define TG_BAUD_115200 115200

#define TG_BAUD_1200 1200 (Baud rate for use with TG_Connect() and TG_SetBaudrate())

#define TG_BAUD_2400 2400

#define TG_BAUD_4800 4800

#define TG_BAUD_57600 57600

#define TG_BAUD_9600 9600

#define TG_DATA_ALPHA1 7

#define TG_DATA_ALPHA2 8

#define TG_DATA_ATTENTION 2

#define TG_DATA_BATTERY 0 (Data types that can be requested from TG_GetValue())

#define TG_DATA_BETA1 9

#define TG_DATA_BETA2 10

#define TG_DATA_BLINK_STRENGTH 37

#define TG_DATA_DELTA 5

#define TG_DATA_GAMMA1 11

#define TG_DATA_GAMMA2 12

#define TG_DATA_MEDITATION 3

#define TG_DATA_POOR_SIGNAL 1

#define TG_DATA_RAW 4

#define TG_DATA_THETA 6

#define TG_MAX_CONNECTION_HANDLES 128
(Maximum number of Connections that can be requested before being required to free one)

#define TG_STREAM_5VRAW 1

#define TG_STREAM_FILE_PACKETS 2

#define TG_STREAM_PACKETS 0
(Data format for use with TG_Connect() and TG_SetDataFormat())

#define THINKGEAR_API extern

#define WINVER 0x0501

1.1.2 Function Documentation

a. TG_Connect()

THINKGEAR_API int TG_Connect (int connectionId, const char* serialPortName, int serialBaudrate, int serialDataFormat)

Connects a ThinkGear Connection, given by connectionId, to a serial communication (COM) port in order to communicate with a ThinkGear module.It is important to check the return value of this function before attempting to use the Connection further for other functions in this API.

Parameters

- connectionId: The ID of the ThinkGear Connection to connect, as obtained from TG_GetNewConnectionId().
- serialPortName: The name of the serial communication (COM) stream port. COM ports on PC Windows systems are named like " (remember that backslashes in strings in most programming languages need to be escaped), while COM ports on Windows Mobile systems are named like 'COM4:' (note the colon at the end). Linux COM ports may be named like '/dev/ttys0'. Refer to the documentation for your particular platform to determine the available COM port names on your system.
- serialBaudrate: The baudrate to use to attempt to communicate on the serial communication port. Select from one of the TG_BAUD_* constants defined above, such as TG_BAUD_9600 or TG_BAUD_57600.
- serialDataFormat: The type of ThinkGear data stream. Select from one of the TG_STREAM_* constants defined above. Most applications should use TG_STREAM_PACKETS (the data format for Embedded ThinkGear). TG_STREAM_5VRAW is supported only for legacy non-embedded purposes.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if serialPortName could not be opened as a serial communication port for any reason. Check that the name is a valid COM port on your system.
- -3 if serialBaudrate is not a valid TG_BAUD_* value.
- -4 if serialDataFormat is not a valid TG_STREAM_* type.
- 0 on success.

b. TG_Disconnect()

THINKGEAR_API void TG_Disconnect (int connectionId)

Disconnects the ThinkGear Connection, given by connectionId, from its serial communication (COM) port. Note that after this call, the Connection will not be valid to use with any of the API functions that require a valid ThinkGear Connection, except TG_SetStreamLog(), TG_SetDataLog(), TG_Connect(), and TG_FreeConnection(). Note that TG_FreeConnection() will automatically disconnect a Connection as well, so it is not necessary to call this function unless you wish to reuse the connectionId to call TG_Connect() again.

Parameters

- connectionId: The ID of the ThinkGear Connection to disconnect, as obtained from TG_GetNewConnectionId().

c. TG_EnableAutoRead()

THINKGEAR_API int TG_EnableAutoRead(int connectionId, int enable)

Enables or disables background auto-reading of the connection. This has the following implications:

- Setting enabled to anything other than 0 will enable background auto-reading on the specified connection. Setting enabled to 0 will disable it.
- Enabling causes a background thread to be spawned for the connection (only if one was not already previously spawned), which continuously calls TG_ReadPacket(connectionId, -1) at 1ms intervals.
- Disabling will kill the background thread for the connection.
- While background auto-reading is enabled, the calling program can use TG_GetValue() at any time to get the most-recently-received value of any data type. The calling program will have no way of knowing when a value has been updated. For most data types other than raw wave value, this is not much of a problem if the program simply polls TG_GetValue() once a second or so.
- The current implementation of this function will not include proper data synchronization. This means it is possible for a value to be read (by TG_GetValue()) at the same time it is being written to by the background thread, resulting in a corrupted value being read. However, this is extremely unlikely for most data types other than raw wave value.
- While background auto-reading is enabled, the TG_GetValueStatus() function is pretty much useless. Also, the TG_ReadPackets() function should probably not be called.

Parameters

- connectionId: The connection to enable/disable background auto-reading on.
- enable: Zero (0) to disable background auto-reading, any other value to enable.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if unable to enable background auto-reading.
- -3 if an error occurs while attempting to disable background auto-reading.
- 0 on success.

d. TG_EnableBlinkDetection()

THINKGEAR_API int TG_EnableBlinkDetection (int connectionId, int enable)

Enables and disables the non-embedded eye blink detection. Non-embedded blink detection is disabled by default.

NOTE: The constants and thresholds for the eye blink detection is adjusted for TGAM1 only.

Parameters

- connectionId: The ID of the ThinkGear Connection to enable low pass filtering for, as obtained from TG_GetNewConnectionId().
- enable: Enables or disables the non-embedded eye blink detection.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- 0 on success.

e. TG_EnableLowPassFilter()

THINKGEAR_API int TG_EnableLowPassFilter(int connectionId, int rawSamplingRate)

As a ThinkGear Connection reads and parses raw EEG wave values (via the TG_ReadPackets() function), the driver can automatically apply a 30Hz low pass filter to the raw wave data.

Subsequent calls to TG_GetValue() on TG_DATA_RAW will therefore return the filtered value. This is sometimes useful for visualizing (displaying) the raw wave when the ThinkGear is in an electrically noisy environment. This function only applies the filtering within the driver and does not affect the behavior of the ThinkGear hardware in any way. This function may be called at any time after calling TG_GetNewConnectionId().

Automatic low pass filtering is disabled by default.

NOTE: Automatic low pass filtering is currently only applicable to ThinkGear hardware that samples raw wave at 512Hz (such as TGAM in MindSet). It is not applicable to hardware that samples at 128Hz or 256Hz and should NOT be enabled for those hardware.

Parameters

- connectionId: The ID of the ThinkGear Connection to enable low pass filtering for, as obtained from TG_GetNewConnectionId().
- rawSamplingRate: Specify the sampling rate that the hardware is currently sampling at. Set this to 0 (zero) or any invalid rate at any time to immediately disable the driver's automatic low
- pass filtering.

NOTE: Currently, the only valid raw sampling rate is 512 (MindSet headsets). All other rates will return -2.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if rawSamplingRate is not a valid rate. Currently only a sampling rate of 512 (Hz) is valid (which is the raw sampling rate of MindSet headsets.
- 0 on success.

f. TG_FreeConnection()

THINKGEAR_API void TG_FreeConnection(int connectionId)

Frees all memory associated with the given ThinkGear Connection.

Note that this function will automatically call TG_Disconnect() to disconnect the Connection first, if appropriate, so that it is not necessary to explicitly call TG_Disconnect() at all, unless you wish to reuse the connectionId without freeing it first for whatever reason.

Parameters

- connectionId: The ID of the ThinkGear Connection to disconnect, as obtained from TG_GetNewConnectionId().

g. TG_GetDriverVersion()

THINKGEAR_API int TG_GetDriverVersion()

Returns a number indicating the version of the ThinkGear Communications Driver (TGCD) library accessed by this API.

Useful for debugging version-related issues.

Returns

- The TGCD library's version number.

h. TG_GetNewConnectionId()

THINKGEAR_API int TG_GetNewConnectionId()

Returns an ID handle (an int) to a newly-allocated ThinkGear Connection object.

The Connection is used to perform all other operations of this API, so the ID handle is passed as the first argument to all functions of this API.

When the ThinkGear Connection is no longer needed, be sure to call TG_FreeConnection() on the ID handle to free its resources. No more than TG_MAX_CONNECTION_HANDLES Connection handles may exist simultaneously without being freed.

Returns

- -1 if too many Connections have been created without being freed by TG_FreeConnection().
- -2 if there is not enough free memory to allocate to a new ThinkGear Connection.

The ID handle of a newly-allocated ThinkGear Connection.

i. TG_GetValue()

THINKGEAR_API float TG_GetValue(int connectionId, int dataType)

Returns the most recently read value of the given dataType, which is one of the TG_DATA_* constants defined above.

Use TG_ReadPackets() to read more Packets in order to obtain updated values. Afterwards, use TG_GetValueStatus() to check if a call to TG_ReadPackets() actually updated a particular dataType.

NOTE: This function will terminate the program with a message printed to stderr if connectionId is not a valid ThinkGear Connection, or if dataType is not a valid TG_DATA_* constant.

Parameters

- connectionId: The ID of the ThinkGear Connection to get a data value from, as obtained from TG_GetNewConnectionId().
- datatype: The type of data value desired. Select from one of the TG_DATA_* constants defined above. Although many types of TG_DATA_* constants are available, each model of ThinkGear hardware and headset will only output a certain subset of these data types. Refer to the Communication Protocol document for your particular ThinkGear hardware or headset to determine which data types are actually output by that hardware or headset. Data types that are

not output by the headset will always return their default value of 0.0 when this function is called.

Returns

- The most recent value of the requested dataType.

j. TG_GetValueStatus()

THINKGEAR_API int TG_GetValueStatus (int connectionId, int dataType)

Return

- Non-zero if the dataType was updated by the most recent call to TG_ReadPackets() / TG_GetValue().
- Returns 0 otherwise.

Parameters

- connectionId: The ID of the ThinkGear Connection to get a data value from, as obtained from TG_GetNewConnectionId().
- datatype: The type of data value desired. Select from one of the TG_DATA_* constants defined above.

NOTE: This function will terminate the program with a message printed to stderr if connectionId is not a valid ThinkGear Connection, or if dataType is not a valid TG_DATA_* constant.

k. TG_ReadPackets()

THINKGEAR_API int TG_ReadPackets(int connectionId, int numPackets)

Attempts to use the ThinkGear Connection, given by connectionId, to read numPackets of data from the serial stream.

The Connection will (internally) "remember" the most recent value it has seen of each possible ThinkGear data type, so that any subsequent call to TG_GetValue() will return the most recently seen values.

Set numPackets to -1 to attempt to read all Packets of data that may be currently available on the serial stream.

Note that different models of ThinkGear hardware and headsets may output different types of data values at different rates. Refer to the "Communications Protocol" document for your particular headset to determine the rate at which you need to call this function.

Parameters

- connectionId: The ID of the ThinkGear Connection which should read packets from its serial communication stream, as obtained from TG_GetNewConnectionId().
- numPackets: The number of data Packets to attempt to read from the ThinkGear Connection. Only the most recently read value of each data type will be "remembered" by the ThinkGear Connection. Setting this parameter to -1 will attempt to read all currently available Packets that are on the data stream.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if there were not even any bytes available to be read from the Connection's serial communication stream.
- -3 if an I/O error occurs attempting to read from the Connection's serial communication stream.
- The number of Packets that were successfully read and parsed from the Connection.

l. TG_SendByte()

THINKGEAR_API int TG_SendByte(int connectionId, int b)

Sends a byte through the ThinkGear Connection (presumably to a ThinkGear module).

This function is intended for advanced ThinkGear Command Byte operations.

WARNING: Always make sure at least one valid Packet has been read (i.e. through the TG_ReadPackets() function) at some point BEFORE calling this function. This is to ENSURE the Connection is communicating at the right baud rate. Sending Command Byte at the wrong baud rate may put a ThinkGear module into an indeterminate and inoperable state until it is reset by power cycling (turning it off and then on again).

NOTE: After sending a Command Byte that changes a ThinkGear baud rate, you will need to call TG_SetBaudrate() to change the baud rate of your connectionId as well. After such a baud rate change, it is important to check for a valid Packet to be received by TG_ReadPacket() before attempting to send any other Command Bytes, for the same reasons as describe in the WARNING above.

Parameters

- connectionId: The ID of the ThinkGear Connection to send a byte through, as obtained from TG_GetNewConnectionId().
- b: The byte to send through. Note that only the lowest 8-bits of the value will actually be sent through.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if connectionId is connected to an input file stream instead of an actual ThinkGear COM stream (i.e. nowhere to send the byte to).
- -3 if an I/O error occurs attempting to send the byte (i.e. broken stream connection).
- 0 on success.

m. TG_SetBaudrate()

THINKGEAR_API int TG_SetBaudrate(int connectionId, int serialBaudrate)

Attempts to change the baud rate of the ThinkGear Connection, given by connectionId, to serialBaudrate.

This function does not typically need to be called, except after calling TG_SendByte() to send a Command Byte that changes the ThinkGear module's baud rate. See TG_SendByte() for details.

Parameters

- connectionId: The ID of the ThinkGear Connection to send a byte through, as obtained from TG_GetNewConnectionId().
- serialBaudrate: The baudrate to use to attempt to communicate on the serial communication port. Select from one of the TG_BAUD_* constants defined above, such as TG_BAUD_9600 or TG_BAUD_57600. TG_BAUD_57600 is the typical default baud rate for most ThinkGear models.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if serialBaudrate is not a valid TG_BAUD_* value.
- -3 if an error occurs attempting to set the baud rate.
- -4 if connectionId is connected to an input file stream instead of an actual ThinkGear COM stream.
- 0 on success.

n. TG_SetDataFormat()

THINKGEAR_API int TG_SetDataFormat(int connectionId, int serialDataFormat)

Attempts to change the data Packet parsing format used by the ThinkGear Connection, given by connectionId, to serialDataFormat.

This function does not typically need to be called, and is provided only for special testing purposes.

Parameters

- connectionId The ID of the ThinkGear Connection to send a byte through, as obtained from TG_GetNewConnectionId().
- serialDataFormat The type of ThinkGear data stream. Select from one of the TG_STREAM_* constants defined above. Most applications should use TG_STREAM_PACKETS (the data format for Embedded ThinkGear modules). TG_STREAM_5VRAW is supported only for legacy non-embedded purposes.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if serialDataFormat is not a valid TG_STREAM_* type.
- 0 on success.

o. TG_SetDataLog()

THINKGEAR_API int TG_SetDataLog(int connectionId, const char * filename)

As a ThinkGear Connection reads and parses Packets of data from its serial stream, it may log the parsed data into a log file.

This is useful primarily for debugging purposes. Calling this function with a valid filename will turn this feature on. Calling this function with an invalid filename, or with filename set to NULL, will turn this feature off. This function may be called at any time for either purpose on a ThinkGear Connection.

Parameters

- connectionId: The ID of the ThinkGear Connection to enable data logging for, as obtained from TG_GetNewConnectionId().
- filename: The name of the file to use for data logging. Any existing contents of the file will be erased. Set to NULL or an empty string to disable stream logging by the ThinkGear Connection.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if filename could not be opened for writing. You may check errno for the reason.
- 0 on success.

p. TG_SetStreamLog()

THINKGEAR_API int TG_SetStreamLog(int connectionId, const char * filename)

As a ThinkGear Connection reads bytes from its serial stream, it may automatically log those bytes into a log file.

This is useful primarily for debugging purposes. Calling this function with a valid filename will turn this feature on. Calling this function with an invalid filename, or with filename set to NULL, will turn this feature off. This function may be called at any time for either purpose on a ThinkGear Connection.

Parameters

- connectionId: The ID of the ThinkGear Connection to enable stream logging for, as obtained from TG_GetNewConnectionId().
- filename: The name of the file to use for stream logging. Any existing contents of the file will be erased. Set to NULL or an empty string to disable stream logging by the ThinkGear Connection.

Returns

- -1 if connectionId does not refer to a valid ThinkGear Connection ID handle.
- -2 if filename could not be opened for writing. You may check errno for the reason.
- 0 on success.

q. TG_WriteDataLog()

THINKGEAR_API int TG_WriteDataLog(int connectionId, int insertTimestamp, const char * msg)

r. TG_WriteStreamLog()

THINKGEAR_API int TG_WriteStreamLog(int connectionId, int insertTimestamp, const char * msg)

1.1.3 Pointers and explanations for how to use TGCD to get Raw Wave Values from the Brainsense:

- You should always use TG_STREAM_PACKETS for TG_Connect(). TG_STREAM_5VRAW can only be used to parse data from a non-commercial headset. It won't do anything for a MindSet.

- To capture all the raw wave samples at 512Hz, you will need to use TG_ReadPackets() to poll fairly quickly and often (A new Packet should be arriving from the headset to the operating system every 1.95ms). It is possible to poll less frequently, since serial data is buffered by the operating system to a certain degree. You still need to poll quickly
- enough so that the operating system's buffer does not fill up and data is lost.

- To capture every raw wave sample, you want to call TG_ReadPackets(connectionId, 1). Note that we instruct it to read '1' packet, instead of '-1' (all available packets). Using '1', we can ensure that we look at every
- packet. With -1, only the most recently arrived value is available when you call TG_GetValue().

- After calling TG_ReadPackets(), you will want to check its return value (errCode). You should only attempt to process data if it returns 1, indicating that one Packet was successfully read. If no Packets were available to be read, you could Sleep() for 2 ms to save processor before trying again.

- If one Packet was successfully read in step 4., you should then call TG_GetValueStatus(connectionId, TG_DATA_RAW) to check if a new raw wave value was included in the Packet. This is necessary because not every Packet contains a new raw wave value (for example, the Packet that arrives every 1s carrying the new Attention, Meditation, and signal quality values).

- If TG_GetValueStatus() indicates a new raw wave value arrived in step 5., you can then call TG_GetValue(connectionId, TG_DATA_RAW) to get the new raw wave value. You can then print out that raw wave value, save it to a data structure, or write it to a log file immediately.

- It is recommended you perform steps 3. through 6. repeatedly, without Sleep(), until TG_ReadPackets() returns 0 in step 4. This ensures you consume all currently available Packets before sleeping.

1.2 ThinkGear Serial Stream Communication

The ThinkGear Serial Stream SDK document (formerly referred to as the ThinkGear Communications Protocol) defines, in detail, how to communicate with the ThinkGear modules. In particular, it describes:

1. How to parse the serial data stream of bytes to reconstruct the various types of brainwave data sent by the ThinkGear.
2. How to interpret and use the various types of brainwave data that are sent from the ThinkGear (including Attention, Meditation, and signal quality data) in a BCI application.
3. How to send reconfiguration Command Bytes to the ThinkGear, for on-the-fly customization of the module's behavior and output.

1.2.1. ThinkGear Data Values:

a. POOR_SIGNAL Quality

This unsigned one-byte integer value describes how poor the signal measured by the ThinkGear is. It ranges in value from 0 to 255. Any non-zero value indicates that some sort of noise contamination is detected. The higher the number, the more noise is detected. A value of 200 has a special meaning, specifically that the ThinkGear electrodes aren't contacting a person's skin.

This value is typically output every second, and indicates the poorness of the most recent measurements.

Poor signal may be caused by a number of different things. In order of severity, they are:

- Sensor, ground, or reference electrodes not being on a person's head (i.e. when nobody is wearing the ThinkGear).
- Poor contact of the sensor, ground, or reference electrodes to a person's skin (i.e. hair in the way, or headset which does not properly fit a person's head, or headset not properly placed on the head).
- Excessive motion of the wearer (i.e. moving head or body excessively, jostling the headset).
- Excessive environmental electrostatic noise (some environments have strong electric signals or static electricity buildup in the person wearing the sensor).
- Excessive non-EEG biometric noise (i.e. EMG, EKG/ECG, EOG, etc)

A certain amount of noise is unavoidable in normal usage of ThinkGear and eSense™ algorithm have been designed to detect, correct, compensate for, account for, and tolerate many types of non-EEG noise. Most typical users who are only interested in using the eSense values, such as Attention and Meditation, do not need to worry too much about the POOR_SIGNAL Quality value, except to note that the Attention and Meditation values will not be updated while POOR_SIGNAL is detected. The POOR_SIGNAL Quality value is more useful to some applications which need to be more sensitive to noise (such as some medical or research applications), or applications which need to know right away when there is even minor noise detected.

b. eSense Meters

For all the different types of eSenses (i.e. Attention, Meditation), the meter value is reported on a relative eSense scale of 1 to 100. On this scale, a value between 40 to 60 at any given moment in time is considered “neutral”, and is similar in notion to “baselines” that are established in conventional EEG measurement techniques (though the method for determining a ThinkGear baseline is proprietary and may differ from conventional EEG). A value from 60 to 80 is considered “slightly elevated”, and may be interpreted as levels being possibly higher than normal (levels of Attention or Meditation that may be higher than normal for a given person). Values from 80 to 100 are considered “elevated”, meaning they are strongly indicative of heightened levels of that eSense.

Similarly, on the other end of the scale, a value between 20 to 40 indicates “reduced” levels of the eSense, while a value between 1 to 20 indicates “strongly lowered” levels of the eSense. These levels may indicate states of distraction, agitation, or abnormality, according to the opposite of each eSense.

An eSense meter value of 0 is a special value indicating the ThinkGear is unable to calculate an eSense level with a reasonable amount of reliability. This may be (and usually is) due to excessive noise as described in the POOR_SIGNAL Quality section above.

The reason for the somewhat wide ranges for each interpretation is that some parts of the eSense algorithm are dynamically learning, and at times employ some “slow-adaptive” algorithms to adjust to natural fluctuations and trends of each user, accounting for and compensating for the fact that EEG in the human brain is subject to normal ranges of variance and fluctuation. This is part of the reason why ThinkGear sensors are able to operate on a wide range of individuals under an extremely wide range of personal and environmental conditions while still giving good accuracy and reliability. Developers are encouraged to further interpret and adapt these guideline ranges to be fine-tuned for their application (as one example, an application could disregard values below 60 and only react to values between 60-100, interpreting them as the onset of heightened attention levels).

c. ATTENTION eSense

This unsigned one-byte value reports the current eSense Attention meter of the user, which indicates the intensity of a user's level of mental “focus” or “attention”, such as that which occurs during intense concentration and directed (but stable) mental activity. Its value ranges from 0 to 100. Distractions, wandering thoughts, lack of focus, or anxiety may lower the Attention meter levels. See eSense Meters above for details about interpreting eSense levels in general.

By default, output of this Data Value is enabled. It is typically output once a second.

d. MEDITATION eSense

This unsigned one-byte value reports the current eSense Meditation meter of the user, which indicates the level of a user's mental “calmness” or “relaxation”. Its value ranges from 0 to 100. Note that Meditation is a measure of a person's mental levels, not physical levels, so simply relaxing all the muscles of the body may not immediately result in a heightened Meditation level. However, for most people in most normal circumstances, relaxing the body often helps

the mind to relax as well. Meditation is related to reduced activity by the active mental processes in the brain, and it has long been an observed effect that closing one's eyes turns off the mental activities which process images from the eyes, so closing the eyes is often an effective method for increasing the Meditation meter level. Distractions, wandering thoughts, anxiety, agitation, and sensory stimuli may lower the Meditation meter levels. See “eSense Meters” above for details about interpreting eSense levels in general.

By default, output of this Data Value is enabled. It is typically output once a second.

e. 8BIT_RAW Wave Value

This unsigned one-byte value is equivalent to the signed RAW Wave Value (16-bit) described below, except that it is scaled to be unsigned, and only the most significant 8 bits are output (where “most significant” is defined based on the specific ThinkGear hardware). This makes it possible to output raw wave values given the bandwidth restrictions of serial communications at a 9600 baud rate, at the cost of not outputting the lowest couple bits of precision. For many applications (such as realtime display of the graph of the raw wave), showing 8 bits of precision is sufficient, since the human eye typically cannot rapidly discern pixels which may correspond to the lower bits of precision anyways. If more precision is required, consider using the normal signed RAW Wave Value (16-bit) (described below) output at a higher baud rate.

Although only the most significant 8 bits are output when 8BIT_RAW output is enabled, all calculations are still performed within the ThinkGear based on the maximum precision of raw wave information available to the ThinkGear hardware, so no information is discarded internally. Only the outputted raw value is reduced to 8 bits, to save serial bandwidth.

By default, output of this Data Value is disabled. Like the regular signed 16-bit RAW Wave Value, the 8BIT_RAW Wave Value is typically output 128 times a second, or approximately once every 7.8 ms.

This Data Value is only available in ThinkGear Embedded modules (TGEM). It is not available from ThinkGear ASIC (TGAT/TGAM1), which is used in MindWave, MindWave Mobile and Brainsense.

f. RAW_MARKER Section Start

This is not really a Data Value, and is primarily only useful for debugging very precise timing and synchronization of the raw wave, or research purposes. Currently, the value will always be 0x00.

By default, output of this Data Value is disabled. It is typically output once a second.

This Data Value is only available in ThinkGear modules. It is not available from ThinkGear ASIC (such as MindWave, MindWave Mobile and Brainsense).

g. RAW Wave Value (16-bit)

This Data Value consists of two bytes, and represents a single raw wave sample. Its value is a signed 16-bit integer that ranges from -32768 to 32767. The first byte of the Value represents the high-order bits of the twos-compliment value, while the second byte represents the low-

order bits. To reconstruct the full raw wave value, simply shift the first byte left by 8 bits, and bitwise-or with the second byte:

short raw = (Value[0]<<8) | Value[1];
where Value[0] is the high-order byte, and Value[1] is the low-order byte.

In systems or languages where bit operations are inconvenient, the following arithmetic operations may be substituted instead:

raw = Value[0]*256 + Value[1];
if(raw >= 32768) raw = raw - 65536;
where raw is of any signed number type in the language that can represent all the numbers from -32768 to 32767.

However, in Java you use the following code instead in order to avoid problems when implementing the parsing of CODE 0x80 (Raw Wave Data). This is due to the fact that Java has no unsigned data types, which has to be handled carefully when combining the two bytes.

```
private int[]   highlow = new int[2];
highlow[0] = (int)(payload[index] & 0xFF);
highlow[1] = (int)(payload[index+1] & 0xFF);
raw = (highlow[0] * 256) + highlow[1];
if( raw > 32768 ) raw -= 65536;
```

Each ThinkGear model reports its raw wave information in only certain areas of the full -32768 to 32767 range. For example, ThinkGear ASIC may only report raw waves that fall between approximately -2048 to 2047, while ThinkGear modules may only report raw waves that fall between approximately 0 to 1023. Please consult the documentation for your particular ThinkGear hardware for more information.

For TGEM the output of this Data Value is disabled by default. When enabled, the RAW Wave Value is typically outputted 128 times a second, or approximately once every 7.8ms. The ThinkGear ASIC (TGAT/TGAM1), however, outputs this value 512 times a second, or approximately once every 2ms. And it is usually turned on by default. The default can be adjusted at the module level, so please consult the spec sheet for more details.

Because of the high rate at which this value is output, and the number of bytes of data involved, it is only possible to output the 16-bit RAW Wave Value on the serial communication stream at 57,600 baud and above. If raw wave information at 9600 baud is desired, consider the 8BIT_RAW Wave Value output instead (described above).

h. EEG_POWER

This Data Value represents the current magnitude of 8 commonly-recognized types of EEG frequency bands (brainwaves). It consists of eight 4-byte floating point numbers in the following order: delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low-alpha (7.5 - 9.25Hz), high-alpha (10 - 11.75Hz), low-beta (13 - 16.75Hz), high-beta (18 - 29.75Hz), low-gamma (31 - 39.75Hz), and mid-gamma (41 - 49.75Hz). These values have no units and therefore are only meaningful when compared to each other and to themselves, for considering relative quantity and temporal

fluctuations. The floating point format is standard big-endian IEEE 754, so the 32 bytes of the Values can therefore be directly cast as a float* in C (on big-endian environments) to be used as an array of floats.

By default, output of this Data Value is disabled. When enabled, it is typically output once a second.

This version of EEG_POWER, using floating point numbers, is only available in ThinkGear modules, and not in ASIC. For the ASIC equivalent, see ASIC_EEG_POWER_INT. As of ThinkGear Firmware v1.7.8, the ASIC version is the standard and preferred format for reading EEG band powers, and this floating point format is deprecated to backwards-compatibility purposes only.

i. ASIC_EEG_POWER_INT

This Data Value represents the current magnitude of 8 commonly-recognized types of EEG (brainwaves). It is the ASIC equivalent of EEG_POWER, with the main difference being that this Data Value is output as a series of eight 3-byte unsigned integers instead of 4-byte floating point numbers. These 3-byte unsigned integers are in big-endian format.

The eight EEG powers are output in the following order: delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low-alpha (7.5 - 9.25Hz), high-alpha (10 - 11.75Hz), low-beta (13 - 16.75Hz), high-beta (18 - 29.75Hz), low-gamma (31 - 39.75Hz), and mid-gamma (41 - 49.75Hz). These values have no units and therefore are only meaningful compared to each other and to themselves, to consider relative quantity and temporal fluctuations.

By default, output of this Data Value is enabled, and is typically output once a second.

As of ThinkGear Firmware v1.7.8, this form of EEG_POWER is the standard output format for EEG band powers, while the one described in EEG_POWER is only kept for backwards compatibility and only accessible through command switches. Prior to v1.7.8, the EEG_POWER was the standard.

j. Eye Blink Strength

This data value is currently unavailable via the ThinkGear Serial Stream APIs. It is not directly available as output from any current ThinkGear hardware. For TGCD, see the TG_DATA_BLINK_STRENGTH data type for use with the TG_GetValueStatus() and TG_GetValue() functions.

k. Mind-wandering Level

This unsigned one byte value reports the intensity of the user's Mind-wandering Level. Its value ranges from 0 to 10. A value of 0 means the Level is N/A. A value from 1-10 indicates the Level (with higher values indicating higher levels of Mind-wandering.

1.2.2 ThinkGear Packets

ThinkGear components deliver their digital data as an asynchronous serial stream of bytes. The serial stream must be parsed and interpreted as ThinkGear Packets in order to properly extract and interpret the ThinkGear Data Values described in the chapter above.

A ThinkGear Packet is a packet format consisting of 3 parts:
1.Packet Header 2.Packet Payload 3.Payload Checksum
ThinkGear Packets are used to deliver Data Values (described in the previous chapter) from a ThinkGear module to an arbitrary receiver (a PC, another microprocessor, or any other device that can receive a serial stream of bytes). Since serial I/O programming APIs are different on every platform, operating system, and language, it is outside the scope of this document (see your platform's documentation for serial I/O programming). This chapter will only cover how to interpret the serial stream of bytes into ThinkGear Packets, Payloads, and finally into the meaningful Data Values described in the previous chapter.

The Packet format is designed primarily to be robust and flexible: Combined, the Header and Checksum provide data stream synchronization and data integrity checks, while the format of the Data Payload ensures that new data fields can be added to (or existing data fields removed from) the Packet in the future without breaking any Packet parsers in any existing applications/devices. This means that any application that implements a ThinkGear Packet parser properly will be able to use newer models of ThinkGear modules most likely without having to change their parsers or application at all, even if the newer ThinkGear includes new data fields.

a. Packet Structure:
Packets are sent as an asynchronous serial stream of bytes. The transport medium may be UART, serial COM, USB, bluetooth, file, or any other mechanism which can stream bytes.

Each Packet begins with its Header, followed by its Data Payload, and ends with the Payload's Checksum Byte, as follows:

Fig.1.1. Packet Structure

The [PAYLOAD…] section is allowed to be up to 169 bytes long, while each of [SYNC], [PLENGTH], and [CHKSUM] are a single byte each. This means that a complete, valid Packet is a minimum of 4 bytes long (possible if the Data Payload is zero bytes long, i.e. empty) and a maximum of 173 bytes long (possible if the Data Payload is the maximum 169 bytes long).

b. Procedure for properly parsing ThinkGear Packets:

i. Packet Header

The Header of a Packet consists of 3 bytes: two synchronization [SYNC] bytes (0xAA 0xAA), followed by a [PLENGTH] (Payload length) byte:

[SYNC] [SYNC] [PLENGTH]

The two [SYNC] bytes are used to signal the beginning of a new arriving Packet and are bytes with the value 0xAA (decimal 170). Synchronization is two bytes long, instead of only one, to reduce the chance that [SYNC] (0xAA) bytes occurring within the Packet could be mistaken for the beginning of a Packet. Although it is still possible for two consecutive [SYNC] bytes to appear within a Packet (leading to a parser attempting to begin parsing the middle of a Packet as the beginning of a Packet) the [PLENGTH] and [CHKSUM] combined ensure that such a "mis-sync'd Packet" will never be accidentally interpreted as a valid packet (see Payload Checksum below for more details).

The [PLENGTH] byte indicates the length, in bytes, of the Packet's Data Payload [PAYLOAD…] section, and may be any value from 0 up to 169. Any higher value indicates an error (PLENGTH TOO LARGE). Be sure to note that [PLENGTH] is the length of the Packet's Data Payload, NOT of the entire Packet. The Packet's complete length will always be [PLENGTH] + 4.

ii. Data Payload

The Data Payload of a Packet is simply a series of bytes. The number of Data Payload bytes in the Packet is given by the [PLENGTH] byte from the Packet Header. The interpretation of the Data Payload bytes into the ThinkGear Data Values described in Chapter 1 is defined in detail in the Data Payload Structure section below. Note that parsing of the Data Payload typically should not even be attempted until after the Payload Checksum Byte [CHKSUM] is verified as described in the following section.

iii. Payload Checksum

The [CHKSUM] Byte must be used to verify the integrity of the Packet's Data Payload. The Payload's Checksum is defined as:

- Summing all the bytes of the Packet's Data Payload
- Taking the lowest 8 bits of the sum
- Performing the bit inverse (one's compliment inverse) on those lowest 8 bits

A receiver receiving a Packet must use those 3 steps to calculate the checksum for the Data Payload they received, and then compare it to the [CHKSUM] Checksum Byte received with the Packet. If the calculated payload checksum and received [CHKSUM] values do not match, the entire Packet should be discarded as invalid. If they do match, then the receiver may procede to parse the Data Payload as described in the "Data Payload Structure" section below.

iv. Data Payload Structure

Once the Checksum of a Packet has been verified, the bytes of the Data Payload can be parsed. The Data Payload itself consists of a continuous series of Data Values, each contained in a series of bytes called a DataRow. Each DataRow contains information about what the Data Value represents, the length of the Data Value, and the bytes of the Data Value itself. Therefore, to parse a Data Payload, one must parse each DataRow from it until all bytes of the Data Payload have been parsed.

v. DataRow Format

A DataRow consists of bytes in the following format:

```
([EXCODE]...) [CODE]  ([VLENGTH])  [VALUE...]
_____________________ _____________ ___________
^^^^(Value Type)^^^^     ^^(length)^^     ^^(value)^^
```

Bytes in parentheses are conditional, meaning that they only appear in some DataRows, and not in others. See the following description for details.

The DataRow may begin with zero or more [EXCODE] (Extended Code) bytes, which are bytes with the value 0x55. The number of [EXCODE] bytes indicates the Extended Code Level. The Extended Code Level, in turn, is used in conjuction with the [CODE] byte to determine what type of Data Value this DataRow contains. Parsers should therefore always begin parsing a DataRow by counting the number of [EXCODE] (0x55) bytes that appear to determine the Extended Code Level of the DataRow's [CODE].

The [CODE] byte, in conjunction with the Extended Code Level, indicates the type of Data Value encoded in the DataRow. For example, at Extended Code Level 0, a [CODE] of 0x04 indicates that the DataRow contains an eSense Attention value. For a list of defined [CODE] meanings, see the CODE Definitions Table below. Note that the [EXCODE] byte of 0x55 will never be used as a [CODE] (incidentally, the [SYNC] byte of 0xAA will never be used as a [CODE] either).

If the [CODE] byte is between 0x00 and 0x7F, then the the [VALUE…] is implied to be 1 byte long (referred to as a Single-Byte Value). In this case, there is no [VLENGTH] byte, so the single [VALUE] byte will appear immediately after the [CODE] byte.

If, however, the [CODE] is greater than 0x7F, then a [VLENGTH] (“Value Length”) byte immediately follows the [CODE] byte, and this is the number of bytes in [VALUE…] (referred to as a Multi-Byte Value). These higher CODEs are useful for transmitting arrays of values, or values that cannot be fit into a single byte.

The DataRow format is defined in this way so that any properly implemented parser will not break in the future if new CODEs representing arbitrarily long DATA… values are added (they simply ignore unrecognized CODEs, but do not break in parsing), the order of CODEs is rearranged in the Packet, or if some CODEs are not always transmitted in every Packet.

A procedure for properly parsing Packets and DataRows is given below in Step-By-Step Guide to Parsing a Packet and Step-By-Step Guide to Parsing DataRows in a Packet Payload, respectively.

CODE Definitions Table

Single-Byte CODEs

Extended Code Level	[CODE]	[LENGTH]	Data Value Meaning
0	0x02	-	POOR_SIGNAL Quality (0-255)
0	0x03	-	HEART_RATE (0-255) Once/s on EGO.
0	0x04	-	ATTENTION eSense (0 to 100)
0	0x05	-	MEDITATION eSense (0 to 100)
0	0x06	-	8BIT_RAW Wave Value (0-255)
0	0x07	-	RAW_MARKER Section Start (0)

Multi-Byte CODEs

Extended Code Level	[CODE]	[LENGTH]	Data Value Meaning
0	0x80	2	RAW Wave Value: a single big-endian. 16-bit two's-compliment signed value (high-order byte followed by low-order byte) (-32768 to 32767)
0	0x81	32	EEG_POWER: eight big-endian 4-byte IEEE 754 floating point values representing delta, theta, low-alpha high-alpha, low-beta, high-beta, low-gamma, and mid-gamma EEG band power values
0	0x83	24	ASIC_EEG_POWER: eight big-endian 3-byte unsigned integer values representing delta, theta, low-alpha high-alpha, low-beta, high-beta, low-gamma, and mid-gamma EEG band power values
0	0x86	2	RRINTERVAL: two byte big-endian unsigned integer representing the milliseconds between two R-peaks
Any	0x55	-	NEVER USED (reserved for [EXCODE])
Any	0xAA	-	NEVER USED (reserved for [SYNC])

Whenever a ThinkGear module is powered on, it will always start in a standard configuration in which only some of the Data Values listed above will be output by default. To enable or disable the types of Data Values output by the ThinkGear, refer to the advanced chapter ThinkGear Command Bytes.

c. Example Packet

The following is a typical Packet. Aside from the [SYNC], [PLENGTH], and [CHKSUM] bytes, all the other bytes (bytes [3] to [10]) are part of the Packet's Data Payload. Note that the DataRows within the Payload are not guaranteed to appear in every Packet, nor are any DataRows that do appear guaranteed by the Packet specification to appear in any particular order. Notation: byte: value // Explanation

[0]: 0xAA // [SYNC]
[1]: 0xAA // [SYNC]
[2]: 0x08 // [PLENGTH] (payload length) of 8 bytes
[3]: 0x02 // [CODE] POOR_SIGNAL Quality
[4]: 0x20 // Some poor signal detected (32/255)
[5]: 0x01 // [CODE] BATTERY Level
[6]: 0x7E // Almost full 3V of battery (126/127)
[7]: 0x04 // [CODE] ATTENTION eSense
[8]: 0x12 // eSense Attention level of 18%
[9]: 0x05 // [CODE] MEDITATION eSense
[10]: 0x60 // eSense Meditation level of 96%
[11]: 0xE3 // [CHKSUM] (1's comp inverse of 8-bit Payload sum of 0x1C)

d. Step-By-Step Guide to Parsing a Packet

- Keep reading bytes from the stream until a [SYNC] byte (0xAA) is encountered.
- Read the next byte and ensure it is also a [SYNC] byte, If not a [SYNC] byte, return to step 1. Otherwise, continue to step 3.
- Read the next byte from the stream as the [PLENGTH]. If [PLENGTH] is 170 ([SYNC]), then repeat step 3. If [PLENGTH] is greater than 170, then return to step 1 (PLENGTH TOO LARGE). Otherwise, continue to step 4.
- Read the next [PLENGTH] bytes of the [PAYLOAD…] from the stream, saving them into a storage area (such as an unsigned char payload[256] array). Sum up each byte as it is read by incrementing a checksum accumulator (checksum += byte).
- Take the lowest 8 bits of the checksum accumulator and invert them. Here is the C code:
  checksum &= 0xFF;
  checksum = ~checksum & 0xFF;
- Read the next byte from the stream as the[CHKSUM] byte. If the [CHKSUM] does not match your calculated chksum (CHKSUM FAILED). Otherwise, you may now parse the contents of the Payload into DataRows to obtain the Data Values, as described below. In either case, return to step 1.

e. Step-By-Step Guide to Parsing DataRows in a Packet Payload

Repeat the following steps for parsing a DataRow until all bytes in the payload[] array ([PLENGTH] bytes) have been considered and parsed:

- Parse and count the number of [EXCODE] (0x55) bytes that may be at the beginning of the current DataRow.
- Parse the [CODE] byte for the current DataRow.
- If [CODE] >= 0x80, parse the next byte as the [VLENGTH] byte for the current DataRow.
- Parse and handle the [VALUE…] byte(s) of the current DataRow, based on the DataRow's [EXCODE] level, [CODE], and [VLENGTH].
- If not all bytes have been parsed from the payload[] array, return to step 1. to continue parsing the next DataRow.

1.2.3 Command Byte Syntax

A Command Byte is formed by 8 bits, each of which are either set or unset. The lowest (least significant) four bits are used to control modes, such as 9600 vs. 57.6k baud mode, attention output enabled or disabled mode, etc. The upper (most significant) four bits, known as the "Command Page", define the meaning of the lower four bits.

For example, a Command Byte of 0x0E has the bit pattern of 0000 1110. The upper (most significant) four bits are 0000, which refers to Command Page zero. Looking on the table below for Command Page zero (for 1.6 firmware), we see that the lower four bits of 1110 are used to control the settings for baudrate, raw wave output, meditation output, and attention output, respectively. Because the baudrate bit is 1, the baudrate of the module will be set to 57.6k mode. Because the raw wave output and meditation output bits are each 1, each of those types of output will be enabled. Because the attention output bit is 0, attention output becomes disabled. Hence, sending a byte of 0x0E to the ThinkGear module instructs it to operate at 57.6k baud mode with raw and meditation values output in packets, but no attention values.

Please note that the ordering of the Command Pages significantly changed between versions 1.6 and 1.7 of the module firmware. Also note that ThinkGear ASIC (such as MindWave and MindWave Mobile) only recognize the 4 Command Bytes on Page 0 for 1.7 firmware; all other Command Bytes may put the ThinkGear ASIC into an inoperable state until it is power-cycled.

A. Firmware 1.6 Command Byte Table

Page 0 (0000____) (0x0_): **
- bit[0] (____0001): Set/unset to enable/disable attention output
- bit[1] (____0010): Set/unset to enable/disable meditation output
- bit[2] (____0100): Set/unset to enable/disable raw wave output
- bit[3] (____1000): Set/unset to use 57.6k/9600 baud rate

Page 1 (0001____) (0x1_):
- bit[0] (____0001): Set/unset to enable/disable EEG powers output
- bit[1] (____0010): Set/unset to use 10-bit/8-bit raw wave output

Page 15 (1111____) (0xF_):
- bit[0] (____0001): Set/unset to enable/disable Testmode

**: After sending this Page byte, the application itself must change itself to begin communicating at the new requested baud rate, and then wait at that new requested baud rate for a complete, valid Packet to be received from the ThinkGear before attempting to send any other command bytes to the ThinkGear. Sending another command byte before performing this check may put the ThinkGear module into an indeterminate (and inoperable) state until it is power-cycled.

B. Firmware 1.7 Command Byte Table

ThinkGear ASIC only recognizes Command Bytes from Page 0 below. Any other Command Bytes may put it into an inoperable state until it is power cycled.

Page 0 (0000____) (0x0_): STANDARD/ASIC CONFIG COMMANDS* **
00000000 (0x00): 9600 baud, normal output mode
00000001 (0x01): 1200 baud, normal output mode
00000010 (0x02): 57.6k baud, normal+raw output mode
00000011 (0x03): 57.6k baud, FFT output mode
Page 1 (0001____) (0x1_): RAW WAVE OUTPUT
bit[0] (____0001): Set/unset to enable/disable raw wave output
bit[1] (____0010): Set/unset to use 10-bit/8-bit raw wave output
bit[2] (____0100): Set/unset to enable/disable raw marker output
bit[3] (____1000): Ignored
Page 2 (0010____) (0x2_): MEASUREMENTS OUTPUTS
bit[0] (____0001): Set/unset to enable/disable poor quality output
bit[1] (____0010): Set/unset to enable/disable EEG powers (int) output
bit[2] (____0100): Set/unset to enable/disable EEG powers (legacy/floats) output
bit[3] (____1000): Set/unset to enable/disable battery output***
Page 3 (0011____) (0x3_): ESENSE OUTPUTS
bit[0] (____0001): Set/unset to enable/disable attention output
bit[1] (____0010): Set/unset to enable/disable meditation output
bit[2] (____0100): Ignored
bit[3] (____1000): Ignored
Page 6 (0110____) (0x6_): BAUD RATE SELECTION* **
01100000 (0x60): No change
01100001 (0x61): 1200 baud
01100010 (0x62): 9600 baud
01100011 (0x63): 57.6k baud

*: Note that pages 0 and 6 are a little different from most command pages. While most pages use each of the 4 bits as an enable/disable switch commands for separate setting, pages 0 and 6 use the entire command value as a single command.

**: After sending this Page byte, the application itself must change itself to begin communicating at the new requested baud rate, and then wait at that new requested baud rate for a complete, valid Packet to be received from the ThinkGear before attempting to send any other command bytes to the ThinkGear. Sending another command byte before performing this check may put the ThinkGear module into an indeterminate (and inoperable) state until it is power-cycled.

***: Battery level sampling not available on some ThinkGear models.