

# ThinkGear SDK for iOS: API Reference

---

January 11, 2013



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 consumer thresholds 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™, CogniScore™, ThinkGear™, MindSet™, MindWave™, NeuroBoy™, NeuroSky®

**NO WARRANTIES: THE NEUROSKY PRODUCT FAMILIES AND RELATED DOCUMENTATION IS PROVIDED "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 NEUROSKY PRODUCTS OR 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.**

**USAGE OF THE NEUROSKY PRODUCTS IS SUBJECT OF AN END-USER LICENSE AGREEMENT.**

# Contents

<b>TGAccessoryManager Class Reference</b>	<b>4</b>
Overview	4
Tasks	4
Getting the Shared Accessory Manager	4
Setting Up and Tearing Down the Accessory Manager	4
Starting and Stopping the Data Stream	4
Getting Information about the Accessory Manager	4
Getting Information about the Accessory	4
Accessing the Delegate	4
Properties	5
accessory	5
connected	5
delegate	5
dispatchInterval	5
Class Methods	6
sharedTGAccessoryManager	6
Instance Methods	6
setUpManagerWithInterval:	6
setUpManagerWithInterval:forAccessoryType:	6
startStream	7
stopStream	7
teardownManager	7
getVersion	7
Constants	8
Accessory Type	8
 <b>TGAccessoryDelegate Protocol Reference</b>	 <b>9</b>
Overview	9
Tasks	9
Responding to Connection and Disconnection Events	9
Responding to Data Receipt Events	9
Instance Methods	9
accessoryDidConnect:	9
accessoryDidDisconnect	9
dataReceived:	9

# TGAccessoryManager Class Reference

---

## Overview

The `TGAccessoryManager` class handles connections and data transfer between a ThinkGear-enabled accessory and an iOS device.

## Tasks

### Getting the Shared Accessory Manager

- `+ sharedTGAccessoryManager`

### Setting Up and Tearing Down the Accessory Manager

- `- setupManagerWithInterval:`
- `- setupManagerWithInterval:forAccessoryType:`
- `- teardownManager`

### Starting and Stopping the Data Stream

- `- startStream`
- `- stopStream`

### Getting Information about the Accessory Manager

- `dispatchInterval` *property*
- `- getVersion`

### Getting Information about the Accessory

- `accessory` *property*
- `connected` *property*

### Accessing the Delegate

- `delegate` *property*

## Properties

### accessory

An `EAAccessory` object indicating the accessory that the `TGAcessoryManager` has found. (read-only)

```
@property (nonatomic, readonly) EAAccessory * accessory
```

#### Discussion

If no accessory can be seen by the `TGAcessoryManager`, the value of this property will be `nil`. You can query this property on application startup, for example, to display a static `UIView` indicating that a ThinkGear-enabled accessory should be attached.

### connected

A Boolean value indicating whether a data stream is active between the accessory and the iOS-based device. (read-only)

```
@property (nonatomic, readonly) BOOL connected
```

#### Discussion

A `startStream` call will set this property to `YES`, and a `stopStream` call will set this property to `NO`.

**Important:** This property is **not** an indication of whether an accessory is attached or not for that, you should check whether or not the `accessory` property is `nil`.

### delegate

The object that acts as the delegate of the accessory.

```
@property (nonatomic, assign) id<TGAcessoryDelegate> delegate
```

#### Discussion

The delegate receives notifications about changes to the status of the ThinkGear-enabled accessory, as well as data receipt notifications. The delegate must adopt the `TGAcessoryDelegate` protocol.

### dispatchInterval

The interval, in seconds, between each `dataReceived:` notification sent to the delegate.

```
@property (nonatomic, assign) NSTimeInterval dispatchInterval
```

#### Discussion

This property indicates the guaranteed **minimum** interval between each `dataReceived:` notification. Though unlikely, the actual interval between each `dataReceived:` notification may be much higher than the value specified here, so your application should handle those situations gracefully.

**Important:** The recommended interval period is dependent on the hardware connected. See the iOS Development Guide for recommended intervals.

## Class Methods

### sharedTGAcessoryManager

Returns the shared `TGAcessoryManager` object for the iOS-based device.

```
+ (TGAcessoryManager *)sharedTGAcessoryManager
```

#### Return Value

The shared ThinkGear accessory manager object.

#### Discussion

You should always use this method to obtain the ThinkGear accessory manager object, rather than creating an instance directly.

## Instance Methods

### setUpManagerWithInterval:

Sets up the `TGAcessoryManager` instance to receive data from a ThinkGear accessory that connects via Bluetooth or 30-pin connector. This lets the `TGAcessoryManager` know that you are ready to receive notifications related to ThinkGear-enabled accessories.

```
- (void)setUpManagerWithInterval: (NSTimeInterval)dispatchInterval
```

#### Parameters

- *dispatchInterval* — The interval between each `dataReceived:` notification.

#### Discussion

Until `startStream` is called, the delegate will **not** receive any `dataReceived:` notifications; this method **only** performs setup of the accessory manager and starts the accessory connection and disconnection notifications.

A `TGAcessoryManager` instance should **not** be used prior to calling this method.

### setUpManagerWithInterval:forAccessoryType:

Sets up the `TGAcessoryManager` instance to receive data from a ThinkGear accessory of type `accessoryType`. This lets the `TGAcessoryManager` know that you are ready to receive notifications related to ThinkGear-enabled accessories.

```
- (void)setUpManagerWithInterval: (NSTimeInterval)dispatchInterval
forAccessoryType: (TGAcessoryType)accessoryType
```

### Parameters

- *dispatchInterval* — The interval between each `dataReceived:` notification.
- *accessoryType* — The type of ThinkGear dongle accessory to connect to.

### Discussion

Until `startStream` is called, the delegate will **not** receive any `dataReceived:` notifications; this method **only** performs setup of the accessory manager and starts the accessory connection and disconnection notifications.

A `TGAcessoryManager` instance should **not** be used prior to calling this method.

## startStream

Open up a data stream to the accessory. This starts the dispatch of notifications, at a rate specified by `dispatchInterval`.

```
- (void)startStream
```

### Discussion

When you no longer want to receive `dataReceived:` notifications on your delegate object, you should call the matching `stopStream` method.

## stopStream

Close the data stream that is opened to the accessory. This stops the dispatch of `dataReceived:` notifications.

```
- (void)stopStream
```

### Discussion

Calls to this method must be balanced with a preceding call to the `startStream` method.

## teardownManager

Perform teardown of the `TGAcessoryManager` instance.

```
- (void)teardownManager
```

### Discussion

This method call should be balanced with a preceding call. Typically, this method is called when you have no further use for the `TGAcessoryManager` (e.g. when the application quits).

## getVersion

Returns the version number of `TGAcessory`.

```
- (int)getVersion
```

## Constants

### Accessory Type

These values represent the different types of ThinkGear iPhone accessories that the `TGAcessoryManager` instance can receive data from.

```
enum {
    TGAcessoryTypeDongle = 0,
    TGAcessoryTypeSimulated = 2
};
typedef NSUInteger TGAcessoryType;
```

#### Constants

**TGAcessoryTypeDongle** A ThinkGear accessory that connects via Bluetooth or 30-pin connector.

Declared in `TGAcessoryManager.h`

**TGAcessoryTypeSimulated** A simulated ThinkGear dongle accessory that connects via a `local-host` socket connection. This should only be used in the iPhone simulator, and should not be used on actual iOS devices.

Declared in `TGAcessoryManager.h`



# TGAccessoryDelegate Protocol Reference

---

## Overview

The `TGAccessoryDelegate` protocol defines methods for handling accessory event notifications dispatched from a `TGAccessoryManager` object.

## Tasks

### Responding to Connection and Disconnection Events

- - `accessoryDidConnect:`
- - `accessoryDidDisconnect`

### Responding to Data Receipt Events

- - `dataReceived:` *required method*

## Instance Methods

### `accessoryDidConnect:`

Tells the delegate that the specified accessory was connected to the iOS-based device.

- (void) accessoryDidConnect: (EAAccessory \*) accessory

#### Parameters

- `accessory` — The accessory that was connected to the device.

### `accessoryDidDisconnect`

Tells the delegate that the previously-connected accessory was disconnected.

- (void) accessoryDidDisconnect

### `dataReceived:`

Tells the delegate that data was received from the accessory. (required)

- (void) dataReceived: (NSDictionary \*) data

### Parameters

- `data` — The data that was received from the accessory, stored in a `NSDictionary` data structure. See below for a discussion of the contents of this data structure.

### Discussion

The `NSDictionary` that is passed as a parameter into this method uses the following `NSString`s as keys. All values are stored as integers unless otherwise specified:

- `poorSignal` — The level of "poorness" of the raw headset signal. A value of 0 means that the signal is clean, and a value of 200 means that the headset is effectively off the head.
- `eSenseAttention` — The eSense™ Attention level
- `eSenseMeditation` — The eSense Meditation level
- `blinkStrength` — The value reporting the strength of the user's most recent blink
- `raw` — The raw, unprocessed signal coming from the headset
- `bufferedRaw` — The raw data buffered in an `NSArray` since the last `dataReceived:` call
- `eegDelta` — The delta EEG power band
- `eegTheta` — The theta EEG power band
- `eegLowAlpha` — The low-alpha EEG power band
- `eegHighAlpha` — The high-alpha EEG power band
- `eegLowBeta` — The low-beta EEG power band
- `eegHighBeta` — The high-beta EEG power band
- `eegLowGamma` — The low-gamma EEG power band
- `eegHighGamma` — The high-gamma EEG power band

### CardioChip Specific data types

- `poorSignal` — Known as **Sensor Status** for CardioChip. A value of 200 means the sensor has contact with skin while a value of 0 means the sensors have lost contact.
- `raw` — The raw, unprocessed signal coming from the CardioChip
- `heartRate` — Heart rate in beats per minute
- `rrInt` — R-R interval in milliseconds
- `respiration` — Respiration rate in breaths per minute as a float
- `heartAge` — Apparent heart age
- `enegry` — Energy level measurement

There is no guarantee that any specific key-value pair will exist in the `NSDictionary` passed by the notification. Strictly speaking, you should check for `nil` values returned by every `valueForKey:` call. In practice, you can make the assumption that `raw` will be returned on every notification, and that if any one of the other keys exist in the data returned, then the complete set of key-value pairs will exist.

**Important:** The data returned only captures the **most current state** of the headset data, rather than returning all of the data received from the headset. If you desire higher-resolution data (e.g. for the raw signal data), simply set a lower interval period in the `TGAcessoryManager` to receive notifications more quickly.

**Note:** The notification will be received by a thread other than the main thread, so if your implementation of this method explicitly triggers any GUI updates (e.g. `reloadData` on a `UITableView`), be sure to wrap the method call in a `performSelectorOnMainThread: withObject: waitUntilDone:` call.