PLUX iOS Application Programming Interface

Documentation - iOS API

1. Introduction

The PLUX iOS Application Programming Interface brings to iOS applications all the functionalities of PLUX devices.

The class `PXBioPluxManager` is the class responsible to establish a connect with a device and search only for devices of the biosignalplux range, as well receive the information about the state of the connection.

The class `PXDevice` represents a single device and contains methods to write specific commands on the COMMANDS characteristic. Every write method for the commands has a completion block where we can obtain the results about the write made.

The API consists on a .framework file (PluxAPI.framework) containing the API object code.

2. Framework Overview

- **Bundle Identifier:** com.plux.pluxAPI
- **Current Version:** 1.0.2
- **Build:** 1.0.2.00
- **Deployment Target:** 9.3
- **Devices:** Universal
- **Organization:** Plux
- **Class Prefix:** PX

3. Main Objects

3.1. PXBioPluxManager

This class is responsible to search and communicate with devices of the biosignalsplux range, as well as receiving information of their state.

It's also responsible to setup the API to enable the communication between an iOS application and a BLE device.

We can define a log level (Simple or None) where None is the default option. If we want to see the log messages of the API, it's necessary set the log level to Simple.

To enable the communication and receive the response events, it's necessary to set the `PXBioPluxManagerDelegate`.

3.1.1. Methods

`scanDevices()`
Parameters: Void
Description: Starts a new search for devices of the biosignalplux range. If bluetooth is powered off, than API shows a log error.

`stopScanDevices()`
Parameters: Void
Description: Stop the search for new devices

`connectDevice(device: PXDevice)`
Parameters:
[in]
device - the device to establish a new connection
[out]
Void()
Description: Responsible to connect a specific device

disconnectDevice(device: PXDevice)
Parameters:
[in]
device - the device to disconnect
[out]
Void()
Description: Disconnects a device

3.2. PXBioPluxManagerDelegate

This is the delegate of PXBioPluxManager when we can get several notifications about:

- A successful connection
- A successful disconnection
- A failed attempt to connect a device
- Bluetooth powered off
- Bluetooth powered on

3.2.1. Methods

didDiscoverNewDevice(_ device: PXDevice)
Parameters:
[in]
device - the device found
[out]
Void()
Description: Called when a new device is found

didConnectDevice((_ device: PXDevice)
Parameters:
[in]
device – connected device object
[out]
Void()
Description: Called when a new device connection is established with success

didFailToConnectDevice((_ device: PXDevice)
Parameters:

[in]
device – device failled to connect
[out]
Void()
Description: Called when a new device connection failed

didDisconnectDevice((_ device: PXDevice)
Parameters:
[in]
device – disconnect device object
[out]
Void()
Description: Called when a devices disconnects with success

didBluetoothPoweredOff()
Parameters: Void
Description: Called when a bluetooth is powered off

didBluetoothPoweredOn()
Parameters: Void
Description: Called when a bluetooth is powered on

3.3. PXDevice

The PXDevice represents a single device where it is possible to obtain his name and UUID. This class provides methods to write specific commands to the characteristic COMMANDS.

3.3.1. Variables

```
public let deviceName: String
```
Description: Get the device name of the device

```
public let deviceUUID: String
```
Description: Get the device UUID of the device

3.3.2. Methods

startAcquisitionWithBaseFrequency(baseFrequency: Float, sourcesArray: [PXSource], completionBlock: ((result: Bool, pluxFrame: PXPluxFrame?) -> Void)?)

Parameters:
[in]
baseFrequency - sample frequency that is intended to the signal acquisition
sourceArray - acquisition parameters
completionBlock - the response given on a block
result - returns true if command was written with success, false otherwise
pluxFrame - acquisition data
[out]
Void()
Description: Start the acquisition, writing the start command

stopAcquisitionWithCompletionBlock(completionBlock:((result: Bool) -> Void)?)

Parameters:
[in]
completionBlock - the response given on a block
result - returns true if command was written with success, false otherwise

[out]
Void()
Description: Stops the acquisition, writing the stop command

getVersionOfDeviceWithCompletionBlock(completionBlock: ((result: Bool, pluxDevice: PXPluxDevice?) -> Void)?)

Parameters:
[in]
completionBlock - the response given on a block
result - returns true if command was written with success, false otherwise
pluxDevice - information about the pluxDevice

[out]
Void()
Description: Get the version of device, writing the version command

getDescriptionOfDeviceWithCompletionBlock(completionBlock: ((result: Bool, description: String?) -> Void)?)

Parameters:

[in]

completionBlock - the response given on a block

result - returns true if command was written with success, false otherwise

description - the description of the device

[out]

Void()

Description:

Get the description of device, writing the description command

getBatteryEventsOfDeviceWithCompletionBlock(completionBlock: ((result: Bool, pluxEvent: PXBatteryEvent?) -> Void)?)

Parameters:

[in]

completionBlock - the response given on a block

result - returns true if command was written with success, false otherwise

description - the battery level of the device

[out]

Void()

Description:

Receive the battery events of the device, asynchronously.

resetDevice()

Parameters:

[in]

completionBlock - the response given on a block

[out]

Void()

Description:

Writes the reset command

3.4. PXDevice

Object that allows the user to set the device's channel acquisition parameters.

3.4.1. Arguments

- *port* - sensor port number;
- *numberOfBits* - sensor samples resolution
- *channelMask* - sensor channels to acquire in a bitmask

frequencyDivisor - sensor frequency divisor

-

4. How to integrate the Framework

To integrate the PluxAPI.framework into your project (Swift or Objective-C), just follow this simple steps:

- The simple way:
    - Drag the file .framework to the project navigator in Xcode
    - And that’s it

- The hard way:
    - Select your application project in the Project Navigator (blue project icon) to navigate to the target configuration window and select the application target under the "Targets" heading in the sidebar
    - In the tab bar at the top of that window, open the "General" panel
    - Click on the + button under the "Embedded Binaries" section
    - Select Add Other button
    - Search for the PluxAPI.framework and add it to the project
    - And that’s it

5. How to use the Framework

It is possible to integrate the PluxAPI.framework with Objective-C projects or Swift projects. This sections describes how to use the framework in both languages.

5.1. Swift

```
// Initialize the PLUX API

var centralManager = PXBioPluxManager()
self.centralManager.delegate = self
```

```
// Set the log level

self.centralManager.logLevel = .Simple

// or

self.centralManager.logLevel = .None
```

```
// Search for Devices

self.centralManager.scanDevices()
```

```
// Receive information of a new Device (PXBioPluxManagerDelegate method)

func didDiscoverNewDevice(device: PXDevice) {

       print("Found new Device: \(device.deviceName) | \(device.deviceUUID)")
}
```

```
// Connect Device

var deviceSelected: PXDevice = ...
```

```
self.centralManager.connectDevice(self.deviceSelected)
```

```
// Disconnect Device

var deviceSelected: PXDevice = ...

self.centralManager.disconnectDevice(self.deviceSelected)
```

```
// Start Acquisition

// Begin the acquisition with a base frequency of 1000 Hz and a frequency divisor of 100 Hz

var device : PXDevice = ...

let baseFreq : Float = 1000
let sources : [PXSource] = [PXSource(port: 1, numberOfBits: 16, channelMask: 0x01,
frequencyDivisor: 100)]

self.device.startAcquisitionWithBaseFrequency(baseFreq, sourcesArray: sources) { (result,
pluxFrame) in

	if result {
		if let pluxFrame = pluxFrame {

			print("Start Command Result: \(result) PluxFrame:\(pluxFrame)")
		}
	} else {
		print("Start Command Error!")
	}
}
```

```
// Stop Acquisition

// Begin the acquisition with a base frequency of 1000 Hz and a frequency divisor of 100 Hz

var device : PXDevice = ...
```

```
self.device.stopDeviceWithCompletionBlock { (result) -> Void? in

	print("Stop Command Result: \(result)")
}
```

```
// Get Version

var device : PXDevice = ...

self.device.getVersionOfDeviceWithCompletionBlock { (result, pluxDevice) in

	if result {
		if let pluxDevice = pluxDevice {
			print("Version: \(pluxDevice)")
		}
	} else {
		print("Version Command Error!")
	}
}
```

```
// Get Description

var device : PXDevice = ...

self.device.getDescriptionOfDeviceWithCompletionBlock { (result, description) in

	if result {
		if let description = description {
			print("Description: \(description)")
		}
	} else {
		print("Description Command Error!")
	}
}
```

```
// Get Battery Events

var device : PXDevice = ...

self.device.getBatteryEventsOfDeviceWithCompletionBlock { (result, pluxEvent) in

	if result {
		if let pluxEvent = pluxEvent {
			print("pluxEvent: \( pluxEvent.batteryLevel)")
		}
	} else {
		print("pluxEvent Command Error!")
	}
}
```

```
// Reset Device

var deviceSelected: PXDevice = ...

self.deviceSelected.resetDevice()
```

```
// Delegate Methods (PXBioPluxManagerDelegate)

func didConnectDevice() {

	print(“The device is connected with success”)
}

func didFailToConnectDevice() {

	print(“Failed to connect device”)
}

func didDisconnectDevice() {

	print(“Disconnected device with success”)
}
```

5.2. Objective-C

```
// Initialize the PLUX API

PXBioPluxManager *centralManager = [PXBioPluxManager alloc] init];
centralManager.delegate = self
```

```
// Set the log level

centralManager.logLevel = PXLogLevelSimple;

// or

centralManager.logLevel = PXLogLevelNone;
```

```
// Search for Devices

[centralManager scanDevices];
```

```
// Receive information of a new Device (PXBioPluxManagerDelegate method)

- (void)didDiscoverNewDevice:(PXDevice *)device {

       NSLog(@"Found new Device: %@| %@", device.deviceName, device.deviceUUID);
}
```

```
// Connect Device

PXDevice *deviceSelected = ...

[centralManager connectDevice:deviceSelected]
```

```
// Disconnect Device

PXDevice *deviceSelected = ...

[centralManager disconnectDevice:deviceSelected]
```

```
// Start Acquisition

// Begin the acquisition with a base frequency of 1000 Hz and a frequency divisor of 100 Hz

PXDevice *device = ...

CGFloat baseFreq = 1000;

PXSource *source = [[PXSource alloc] initWithPort:1 numberOfBits:16 channelMask:0x01
frequencyDivisor:100];

NSArray *sourcesArray = [[NSArray alloc] initWithObjects:source, nil];

[device startAcquisitionWithBaseFrequency:baseFreq sourcesArray:sourcesArray
completionBlock:^(BOOL result, PXPluxFrame * _Nullable pluxFrame) {

       NSLog(@"Start Command Result: Frame: %@)", pluxFrame);
}];
```

```
// Stop Acquisition

PXDevice *device = …

[device stopAcquisitionWithCompletionBlock:^(BOOL result) {

       // Do Something
}];
```

```
// Get Version

PXDevice *device = ...

[device getVersionOfDeviceWithCompletionBlock:^(BOOL result, PXPluxDevice * _Nullable
pluxDevice) {

       // Do Something
}];
```

```
// Get Description

PXDevice *device = ...

[device getDescriptionOfDeviceWithCompletionBlock:^(BOOL result, NSString * _Nullable
description) {

        // Do Something
}];
```

```
// Get Description

PXDevice *device = ...

[device resetDevice];
```

```
// Delegate Methods (PXBioPluxManagerDelegate)

- (void)didConnectDevice {

        NSLog(@"The device is connected with success")
}

- (void)didFailToConnectDevice {

        NSLog(@"Failed to connect device")
}

- (void)didDisconnectDevice {

        NSLog(@"Disconnected device with success")
}
```