

BlueParrott iOS SDK 4.0.9

Introduction

This document is an introduction to the BlueParrott iOS SDK that allows access to the programmable BlueParrott Button™ on selected BlueParrott headsets.

The BlueParrott Button on the headset has an initial factory setting that allows the user to mute the headset microphone. This BlueParrott Button can be accessed from a mobile application to program alternative functions.

The BlueParrott SDK enables developers to create apps that interact with this BlueParrott Button thus allowing the headset user to access functions of your application by pressing the BlueParrott Button.

SDK Purpose

The purpose of the BlueParrott SDK is to let you, a corporate or independent developer, develop applications that allow the headset user to benefit from alternative functions for the BlueParrott Button found on BlueParrott headsets.

Through use of the BlueParrott SDK you can access the programmable BlueParrott Button. When your mobile application has been published the headset user may then control programmed activities within your mobile application directly from the headset via the BlueParrott Button.

Typical uses of the BlueParrott SDK include integration with:

- Push to Talk and Voice Messaging Applications, where recording is triggered by pressing the BlueParrott Button
- Voice Recognition and custom Voice Assistant applications, where the Voice Assistant is triggered by tapping the BlueParrott Button
- Other Enterprise Applications, where custom features can be triggered based on one or more button event

The BlueParrott Button can be configured to trigger up to five Button Events in your application:

- Press (for example to start a Push to Talk Call)
- Release (for example to end a Push to Talk Call)
- Single Tap (to invoke a voice recognition or other enterprise application)
- Double Tap
- Long Press

You may program each of these events for individual functions within your mobile application.

The SDK can also report changes in the proximity sensor on the headset (where available, not all headsets contain a proximity sensor).

The BlueParrott Button may also be programmed to a number of functions that do not result in events being sent to the application (e.g. Speed Dial).

You may limit your application to operating in the foreground only or you may wish to offer a smoother, more integrated experience for the headset user by extending your application to offer handling of button events even when your application is in the background.

SDK Scope

This document covers BlueParrott iOS SDK v4.0.9.

The BlueParrott SDK provides interfaces to the following functions in compatible BlueParrott devices:

- Configuration of the BlueParrott Button for use with the application.
- Setting the BlueParrott Button into 'SDK Mode', this will result in button events being sent to your application.
- Event Handlers for BlueParrott Button events.
- Configuring the BlueParrott Button to Speed Dial: Dials the specified phone number when the button is pressed.
- Configuring the BlueParrott Button to Mute: Disables the microphone on the headset during a phone call. Pressing the BlueParrott Button again will un-mute the headset.
- Setting the BlueParrott Button to “App Mode” with a specific App ID and App Name. This allows another app using the SDK to receive button events, and if multiple apps are installed that make use of the BlueParrott Button, each can check if they are the currently configured app.
- Setting the BlueParrott Button to a custom mode. Some headsets may offer features that can be accessed through the BlueParrott Button by configuring it to a custom mode. Contact BlueParrott for more details on this topic.

The package includes the following development tools:

- SDK Library.
- Example test application to show connectivity and connected device BlueParrott Button state.
- Notes and reference documentation.

Not in Scope

This document does not cover iOS programming specifics, the developer should access the relevant sites for the development language.

Updates

The BlueParrott SDK will be updated from time to time, please check for updates.

Supported Headsets

The following is a list of devices which have a BlueParrott Button and are supported by the BlueParrott SDK.

Headset
BlueParrott M300-XT
BlueParrott B550-XT
BlueParrott B450-XT Classic
BlueParrott B350-XT
BlueParrott B450-XT
BlueParrott C400-XT

BlueParrott Reveal Pro
BlueParrott S450-XT
BlueParrott C300-XT

Supported Operating Systems

The BlueParrott iOS SDK supports iOS 9.1 and higher.

Getting Started

Including the SDK

The minimum iOS version is 9.1.

Add the BlueParrott SDK to your project by dragging the framework file into your project.

In your project's Target, enable Background Modes and tick 'Uses Bluetooth LE accessories' if you will be using the button while the app is in the background.

In your project's `Info.plist` file, ensure that you set a value for the key "Privacy - Bluetooth Always Usage Description" (`NSBluetoothAlwaysUsageDescription`). The value for this key is displayed to the user when asking permission to use Bluetooth and is required for any app that uses Bluetooth. For more information, see: https://developer.apple.com/documentation/bundleresources/information_property_list/nsbluetoothalwaysusagedescription

The sample code for iOS is based on the current OS version of iOS 15. You, the developer must ensure that syntax is correct at the time of building your project.

Permissions

Enable Bluetooth

The BlueParrott SDK manages the Bluetooth connection but you must manage the enabling of Bluetooth in your code. An approach to this is to alert the application user if Bluetooth is not

available and ask the user to turn Bluetooth on in their handset settings, you must then provide another opportunity to connect.

Use specific permissions

There are many different permissions depending on the function of the application program in question; it is up to the programmer to ensure that all other permissions for your program are included (e.g. microphone).

Connect to the BlueParrott Button

Connect to the BlueParrott Button using methods from the BlueParrott SDK.

The first step is to get a handle to the BlueParrott Button on the BlueParrott Headset.

Now you have the ability to check the connection and connect or disconnect to the headset.

The BlueParrott SDK method `connect` manages the BlueParrott Button connection and ensures that the Bluetooth LE connection is available.

NOTE: It is not advised to call the `connect` method in `viewDidLoad` as the Bluetooth hardware may not yet be available. It is recommended to follow the pattern in the code below, where the handle to the headset is retrieved in `viewDidLoad` (to initialize the headset object and the Bluetooth hardware), then call `connect` at a later point (e.g. on user interaction or after the app is finished initializing).

```
// ViewController.h

#import <BPHeadset.h>
@interface ViewController : UIViewController <BPHeadsetListener>
// ...
@end

// ViewController.m

@implementation ViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    // To instantiate, just use `sharedInstance`:
    headset = [BPHeadset sharedInstance];
    [headset addListener:self];
}
```

```

    // To connect, set a listener that conforms to the BPHeadsetListener
    protocol, then call connect:
    - (IBAction)connectButtonTouched:(id)sender {
        [headset connect];
    }
    @end

```

Listen for Connection Events

Through the BlueParrott SDK your app can get a handle to the BlueParrott Button on the headset, you can then add a BlueParrott Headset Listener.

```

// ViewController.h

#import <BPHeadset.h>
@interface ViewController : UIViewController <BPHeadsetListener>
// ...
@end

// ViewController.m

@implementation ViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    // To instantiate, just use `sharedInstance`:
    headset = [BPHeadset sharedInstance];
    [headset addListener:self];
}

// Headset is connected and mode can be set, but not all
// values have been read yet
- (void) onConnect {
    NSLog(@"Connected");
    // Update UI etc.
}

// Values have been read from headset (e.g. Speed Dial number etc.)
- (void) onValuesRead {
    NSLog(@"Finished reading values");
}

```

```

        // Can now read App Name, Speed Dial number etc.
    }

    // Enterprise Values have been read from headset
    - (void) onEnterpriseValuesRead {
        NSLog(@"Finished reading enterprise values");
        // Can now read enterprise values through getConfigValue
    }
@end

```

The BlueParrott listener allows your app to listen and manage activities associated with establishing a connection to the BlueParrott Button:

- Connected
- Connection Progress
- Connection Failure
- Connected
- Values read from headset

When your app is successfully connected to the BlueParrott Button a connected event is triggered.

Connection Progress

You may retrieve the status of progress during the connection process, as connection may take time it is advisable to keep your user informed of progress.

```

// Method from protocol: BPHeadsetListener
// -----

public void onConnectProgress:(BPConnectProgress) status {
    logStatus(getStatusDescription(progressCode));
}

// Note: Constants from BPConnectProgress are as per table

```

Status	Value	Meaning
BPConnectProgressStarted	0	Connection attempt has started
BPConnectProgressScanning	1	Scanning for BLE services
BPConnectProgressFound	2	The BlueParrott service has been found
BPConnectProgressReading	3	BLE connection established, reading settings from headset

Connection Failure

If the connection attempt fails the following error codes may be returned and should be handled in your code.

```
// Method from protocol: BPHeadsetListener
// -----

- (void) onConnectFailure:(BPConnectError) reasonCode {
    //handle the connection failure here
    //this may include instruction "Retry or turn headset off then on"
}

// Note: Constants from BPConnectError are as per table
```

Reason Code	Value	Meaning
BPConnectErrorUnknown	0	Unknown error occurred

BPConnectErrorBluetoothDisabled	1	Bluetooth is not turned on
BPConnectErrorFirmwareTooOld	2	Firmware on the headset is not offering BlueParrott Button Service. Firmware may be too old
BPConnectErrorSDKTooOld	3	This SDK version is too old to connect to the firmware version on headset

In some situations you may need to advise your user to turn the headset off then on again.

Setting the BlueParrott Button SDK Mode

Once connected to the BlueParrott Button the next step is to enable the BlueParrott Button in order to send events from the BlueParrott Button on the headset to your application on the handset. It is possible to check to see if the BlueParrott Button has been enabled previously and if not you may proceed to enable it.

```
// ViewController.m
// Enable SDK Mode
if (!headset.sdkModeEnabled) {
    [headset enableSDKMode];
}

// Disable SDK Mode
if (headset.sdkModeEnabled) {
    [headset disableSDKMode];
}
```

If required by your program you may also disable the BlueParrott Button SDK. This will reset the BlueParrott Button to the factory setting of a mute button.

enableSDKMode Method

Signature:

enableSDKMode()

enableSDKMode(String appName)

Description:

There are now two methods to enable SDK mode in the SDK: enableSDKMode() and enableSDKMode(String appName). The new method allows the app to set the App Name while still putting the headset into SDK mode. This can allow the current app to know if it was the last app to put the headset into SDK mode.

Callbacks:

Results in a call to onModeUpdate() or onModeUpdateFailure()

setMuteMode Method

Signature:

setMuteMode()

Description:

This sets the headset into the default Mute mode.

Callbacks:

Results in a call to onModeUpdate() or onModeUpdateFailure()

setCustomMode Method

Signature:

setCustomMode(Integer mode)

Description:

This sets the headset into a custom mode.

Callbacks:

Results in a call to onModeUpdate() or onModeUpdateFailure()

Discussion:

The mode can be set to one of several preset modes, or a custom integer more can be set (if supported by the headset). For more details on custom modes, contact your BlueParrott representative.

Preset modes are as follows:

Reason Code	Value	Meaning
BPButtonModeUnknown	-1	An unknown mode. This can be returned if the current mode has not been read yet.
BPButtonModeMute	0	Mute on call mode. This is the default mode.
BPButtonModeSpeedDial	1	Speed dial a specific number.
BPButtonModeApp	2	Compatible App or SDK Mode. (NOTE: In order to put the headset into App Mode or SDK mode, use the specific methods provided - <code>enableSDKMode</code> and <code>setAppModeWithAppKey</code>)

Listen for BlueParrott Button Mode Update

You can monitor the success of enabling the SDK using the mode update methods from the BlueParrott Headset Listener.

```
// Methods from protocol: BPHeadsetListener
// -----

- (void) onModeUpdate {
    NSLog(@"Mode Updated");
}
```

Listen for BlueParrott Button Mode Update Failure

If mode update fails you may look for one of the following update errors to provide feedback to the user.

```
// Methods from protocol: BPHeadsetListener
// -----

- (void) onModeUpdateFailure:(BPMModeUpdateError) reasonCode {
    NSLog(@"Mode Update Failed. Reason %@", [self
getUpdateErrorDescription:reasonCode]);
    //handle error
}
```

Reason Code	Value	Meaning
BPMModeUpdateErrorUnknown	0	Writing to headset over BLE failed
BPMModeUpdateErrorDisconnected	1	BLE connection not available

Once you enable the SDK on the BlueParrott Button data can be sent over Bluetooth from the headset to your app for BlueParrott Button events.

You can then use BlueParrott Headset listener to monitor the traffic from the headset.

Once enabled the SDK remains enabled on the headset until either your app disables the SDK or the user resets the headset.

Listening for BlueParrott Button Events

Once connected to the BlueParrott Button, with the button enabled to send events the BlueParrott Listener for the BlueParrottSDK may be used to monitor clicks of the BlueParrott Button.

When the user clicks on the headset BlueParrott Button your app will receive an event and can react to the users interactions.

The BlueParrott Listener provides methods to allow your code to interact with the users headset BlueParrott Button clicks:

- Button Down - button has been pressed
- Button Up - button has been released
- Tap - single tap
- Double Tap
- Long Press

```
// Methods from protocol: BPHeadsetListener
```

```
// -----
```

```
- (void) onButtonDown:(BPButtonID) buttonID {  
    NSLog(@"Button Down");  
    //your code goes here  
}
```

```
- (void) onButtonUp:(BPButtonID) buttonID {  
    NSLog(@"Button Up");  
    //your code goes here  
}
```

```
- (void) onTap:(BPButtonID) buttonID {  
    NSLog(@"Tap");  
    //your code goes here  
}
```

```
- (void) onDoubleTap:(BPButtonID) buttonID {  
    NSLog(@"Double Tap");  
    //your code goes here  
}
```

```
- (void) onLongPress:(BPButtonID)buttonID {  
    NSLog(@"Long Press");  
    //your code goes here  
}
```

Listening for Proximity Events

Once connected to the BlueParrott Button, the BlueParrottSDK may be used to monitor changes in the proximity sensor on the headset.

Possible values for the proximity state are `BPPProximityStateOff`, `BPPProximityStateOn` and `BPPProximityStateUnknown`.

```
// Methods from protocol: BPHeadsetListener
// -----

- (void) onProximityChanged:(BPPProximityState) proximityState {
    NSLog(@"Proximity State Changed.");
    //your code goes here
}
```

Disconnecting from the BlueParrott Button

Generally at this point you will have established a handle to the BlueParrott Headset Button in order to connect to the BlueParrott Button.

Now you have the ability to check if the BlueParrott Button is connected and you can disconnect the BlueParrott Button in your code if required.

```
// ViewController.m

- (IBAction)disconnectButtonTouched:(id)sender {
    if (headset.connected) {
        [headset disconnect];
    }
}
@end
```

Listen for Disconnect

Using the BlueParrott Headset Listener you may listen for a disconnect event in your program and carry out any appropriate actions and housekeeping at this point.

```

// ViewController.h

#import <BPHeadset.h>
@interface ViewController : UIViewController <BPHeadsetListener>
// ...
@end

// ViewController.m

@implementation ViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    // To instantiate, just use `sharedInstance`:
    headset = [BPHeadset sharedInstance];
    [headset addListener:self];
}

// Called when the headset is disconnected (e.g. out of range)
- (void) onDisconnect {
    // Notify the user
}
@end

```

Enterprise Values

On some BlueParrott headsets, there may be one or more enterprise values which can relate to hardware-specific functions (e.g. the headset may be put into “warehouse mode” which could affect how pairing is managed). There are a number of properties, events and methods related to the enterprise values in the headset.

enterpriseValuesRead Property (Readonly)

Indicates whether the SDK has finished retrieving all enterprise values from the headset. After the app receives the `onConnect()` event, this will be `FALSE`. After the `onEnterpriseValuesRead()` event, this will return `TRUE`.

onEnterpriseValuesRead Event

This is called on the BlueParrott Headset Listener when the SDK has finished retrieving all enterprise values from the headset.

setConfigValue Method

Signature:

setConfigValue(Integer key, String value)

Description:

Sets the value for a specific enterprise key. Details for the keys applicable for a specific headset and the possible values are described below.

getConfigValue Method

Signature:

getConfigValue(Integer key)

Description:

Retrieves a specific enterprise key. Details for the keys applicable for a specific headset and the possible values are described below.

Returns:

Returns a string representing the value of the enterprise key.

getConfigValues Method

Signature:

getConfigValues()

Description:

Retrieves all enterprise keys from the headset. Details for the keys applicable for a specific headset and the possible values are available from BlueParrott.

Returns:

Returns a dictionary containing NSNumber representations of the keys and Strings for the values.

Configuration Key and Values

Only the keys and values described below should be set in an application.

Config Key	Description
1	Voice Control
2	General Headset configuration
3	Warehouse Features

The values that can be set for each key are detailed below.

Key 1 - Voice Control

This is an integer-type value, where specific bits can be set to change the behavior of the voice recognition feature built into the headset.

Bit	Description
5	'Hello Blue Parrott' now triggers phone command instead of 'Say a command' (B550-XT)
6	Disable the 'always listening' triggers
7	Disable 'answer or ignore' on incoming calls

Key 2 - General Headset Configuration

This is an integer-type value, where specific bits can be set to change some high-level behavior of the headset.

Bit	Description
1	Warehouse mode all modes on. If this is set then all Warehouse modes are enabled and it disregards configuration key 3
14	Reserved
15	Disabled Proximity sensor (B550 only)

Key 3 - Warehouse Features

This is an integer-type value, where specific bits can be set to change different features that may be useful in a warehouse usage scenario.

Bit	Description
0	Overrides the friendly name to have the last 4 digits of the Bluetooth mac address added
1	Enables the pairing list to be cleared every time the headset is put into pair mode
2	Disables the 'Cancel' or 'End Call' function of the MFB
3	Disables the 'Establish SLC' event when the MFB is pressed
4	Disables the 'Answer' event when the MFB is pressed

Other BlueParrott Button Functions

After the application has received the `onValuesRead` callback, the application can query the current state of the BlueParrott Button configuration.

Properties

`appKey` Property (Readonly)

The current App Key for the headset (if the headset is in `BPButtonModeApp`), as a String. If the headset is currently in SDK mode, this will return “sdk”. This can be set by an application, and then later queried to check if another app has configured the BlueParrott Button (and then possibly ignore button events).

`appName` Property (Readonly)

The current App Name for the headset (if the headset is in `BPButtonModeApp`), as a String. If the headset is currently in SDK mode, this will return “SDK” unless the app has set the App Name to something different.

`speedDialNumber` Property (Readonly)

The number that will be dialed by the headset when the BlueParrott Button is pressed, if the headset is in `BPButtonModeSpeedDial`, as a String.

`valuesRead` Property (Readonly)

Indicates whether the SDK has finished retrieving all values from the headset. After the app receives the `onConnect()` event, this will be `FALSE`. After the `onValuesRead()` event, this will return `TRUE`.

`friendlyName` Property (Readonly)

Returns the friendly name of the connected headset.

Methods

The SDK can also be used to set the mode of the headset, which can change the behavior of the BlueParrott Button or potentially change the headset function (e.g. using a Custom Mode could change whether the headset resets its pairing list on being connected to a power source).

Some of the methods that can affect the mode of the BlueParrott Button or headset are described below:

setBondable Method

NOTE: Changing the bondable setting on a headset may cause the headset to restart. Use caution when changing this value. Read the detailed description below for more information.

BlueParrott headsets support BLE bonding by default. This means that when a headset is BLE connected for the first time (e.g. when the SDK is used to connect to the headset), the user will be prompted to allow the headset to “Pair” (on iOS, the term “bond” is not used).

Some applications may require control over whether the headset allows BLE bonding or not. In this case, the `setBondable()` method can be used to change the BLE bonding behavior in the headset. Since BLE bonding involves exchanging keys between the handset and the headset, changing this value will clear out connection keys on the headset, which will remove the pairing with the handset, if one exists. This in turn will cause the headset to restart, which means the method should only be called once when the value needs to be changed.

The SDK will not change the value in the headset if it detects that the setting being used is the same as the current setting, which will prevent the headset from restarting unnecessarily. The app can also check the `bondableEnabled` property to retrieve the current state. It is recommended that the app should be written such that it expects the headset to restart when the value is changed, and communicates this to the end user.

If the bondable setting is disabled on a headset, the headset removes its pairing keys for the handset, but the handset may still have a pairing entry for the headset. This should be removed by the end user after the headset restarts, and the headset should be paired again as normal.

Signature:

`setBondable(BOOL enabled)`

Description:

This sets whether the headset is BLE bondable or not. If bondable is disabled by the app, the headset will need to be unpaired and re-paired in order to remove any existing bond with the

phone on the handset. If bonding is enabled on a headset where it was previously disabled, the headset will need to be powered off and then powered on again (so that it reconnects) in order to trigger the bonding.

Callbacks:

Results in a call to `onBondableUpdate()` or `onBondableUpdateFailure()`

setAppMode Method

Signature:

`setAppModeWithAppKey(String appKey, String appName)`

Description:

This sets the headset into App Mode with the given App ID and App Name.

Callbacks:

Results in a call to `onModeUpdate()` or `onModeUpdateFailure()`

setSpeedDialMode Method

Signature:

`setSpeedDialMode(String phoneNumber)`

Description:

This sets the headset into a Speed Dial mode with the given number.

Callbacks:

Results in a call to `onModeUpdate()` or `onModeUpdateFailure()`

Simple Sample Application

The iOS SDK Sample Application is a sample of a simple integration, allowing the developer to get up and running quickly while using the BlueParrott SDK.

The BPSDKSample application runs in the foreground only. It allows you to connect to and disconnect from the BlueParrott Button. You may enable and disable the SDK Mode on the BlueParrott Button as well as enable or disable whether the headset is BLE Bondable. When the SDK is enabled the app gives feedback on the presses of the BlueParrott Button through the log and UI.

This simple demo includes the following features:

- Displays the version of the SDK being used in the app

- Connects to headset BlueParrott Button
- Enables BlueParrott SDK mode
- Utilises BlueParrott listener recognising the BlueParrott Button clicks
- Disables BlueParrott SDK mode
- Disconnects from the headset BlueParrott Button

Below is a sample screen from the application:

