# BlueParrott Android SDK

## Introduction

This document is an introduction to the BlueParrott Android SDK.  The SDK allows integration with the programmable BlueParrott Button™ and other customisation of other features and events on selected BlueParrott headsets.

A full reference for the SDK is included in the installation package.

## SDK Purpose

The purpose of the BlueParrott SDK is to let you, a corporate or independent developer integrate your mobile applications with the BlueParrott Button found on BlueParrott headsets, as well as customizing and leveraging other headset functions (Enterprise configuration, Proximity detection).

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).

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 Android SDK v4.3.01**

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.
- Listening for Proximity change events (on supported headsets), when the user puts the headset on and off their head.

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 Android 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 B550-XT |
| BlueParrott B450-XT Classic |
| BlueParrott B650-XT/S650-XT |
| BlueParrott B350-XT |
| BlueParrott B450-XT |
| BlueParrott C400-XT |
| BlueParrott Reveal Pro |
| BlueParrott S450-XT |

| |
|---|
| BlueParrott C300-XT |
| BlueParrott M300-XT |
| BlueParrott B650-XT/S650-XT |

# Supported Operating Systems

The current BlueParrott Android SDK supports Android 6 and higher.

# Getting Started

## Including the SDK

Add the BlueParrott SDK to your project by importing the ‘blueparrottsdk-release.aar’ file into your project as a module.

Then add a dependency in your projects App Gradle file as follows

```
dependencies {
    compile(name: 'blueparrottsdk-release', ext: 'aar')
```

In addition the following will be required in the project build gradle

```
repositories {
        flatDir {
            dirs 'libs'
        }
```

## How the SDK Connects to the Headset

SDK Communication between the handset and headset is via Bluetooth and can be over one of three connection modes

- Auto
- Classic
- Bluetooth Low Energy (BLE).

(This SDK connection is separate to the normal connection between the handset and headset for audio/call handling which is handled automatically by the operating system.) All headsets support SDK connection via BLE, and most but not all support the SDK connection over Classic Bluetooth.

This can be transparent to the developer, although in certain circumstances you may wish to force the SDK to connect over a Classic Bluetooth connection. On some Android versions this is more stable, and also requires fewer Permissions, making it easier to integrate into Enterprise Applications.

You may choose to force the SDK to connect via Classic or BLE Bluetooth or allow it to automatically select the protocol using the Auto option, which will attempt to connect via Classic Bluetooth first and if that fails will attempt to connect over BLE.

## 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.

### Manifest Permissions

You should add the following permissions to your application Manifest:

```
<!-- Request legacy Bluetooth permissions on older devices. -->
<uses-permission android:name="android.permission.BLUETOOTH"
android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
android:maxSdkVersion="30" />

<!-- Needed to connect on Android 12+ -->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

To support connection over BLE, you must add the following additional permissions to the Manifest:

```
<!-- Needed only if your app looks for and connects over Bluetooth Low
Energy on Android 12+ -->
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
android:usesPermissionFlags="neverForLocation" />

<!-- Legacy permissions to scan over BLE prior to Android 12-->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

### Permission Prompts

Some of these permissions require runtime permission from the user.  You will need to prompt the user to grant access in your application.  For Classic connections, just one permission is required (BLUETOOTH_CONNECT).  For BLE Connections, you will also need to prompt for BLUETOOTH_SCAN (and BLUETOOTH_FINE_LOCATION if targeting devices prior to Android 12)

This example checks for permissions for Classic and BLE connections on all supported versions:

```
public void checkPermissions() {

   List<String> permissionRequest=new ArrayList<String>();

   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {

       isScanPermissionGranted= ContextCompat.checkSelfPermission(this,

Manifest.permission.BLUETOOTH_SCAN)==PackageManager.PERMISSION_GRANTED;
       isConnectPermissionGranted= ContextCompat.checkSelfPermission(this,

Manifest.permission.BLUETOOTH_CONNECT)==PackageManager.PERMISSION_GRANTED;

       if (!isScanPermissionGranted) {
           //ONLY REQUIRED IF USING FORCE BLE OR HEADSET THAT DOES NOT
SUPPORT AT COMMANDS
           permissionRequest.add(Manifest.permission.BLUETOOTH_SCAN);
       }

       if (!isConnectPermissionGranted) {
           permissionRequest.add(Manifest.permission.BLUETOOTH_CONNECT);
       }
```

```
   } else {
       //only required for BLE on 11
       isFineLocationGranted= ContextCompat.checkSelfPermission(this,

Manifest.permission.ACCESS_FINE_LOCATION)==PackageManager.PERMISSION_GRANTE
D;
       if (!isFineLocationGranted) {

permissionRequest.add(Manifest.permission.ACCESS_FINE_LOCATION);
          }
   }

   if (!permissionRequest.isEmpty()){
       mPermissionResultLauncher.launch(permissionRequest.toArray(new
String[0]));
   }

}
```

## Connect to the BlueParrott Button

The BlueParrott SDK connect() method creates a connection from your application to the BlueParrott headset.

As described above, there are a number of methods to connect depending on the headset being used. Auto attempts to connect via classic Bluetooth first and if that fails will attempt to connect over BLE.  The connect() method takes one of the following parameters.

| | |
|---|---|
| CONNECT_METHOD_AUTO | Attempts to connect to SDK via classic connection.If this fails will attempt connection over BLE |
| CONNECT_METHOD_CLASSIC | Attempts to connect to SDK via classic connection |
| CONNECT_METHOD_BLE | Attempts to connect to SDK via BLE connection |

```
import com.blueparrott.blueparrottsdk.BPSdk;
import com.blueparrott.blueparrottsdk.BPHeadset;
```

```
BPHeadset headset;

    // Get a handle to the BlueParrott Button on the headset
    headset = BPSdk.getBPHeadset(this);

    if (!headset.connected()) {
                Log.d(TAG, "Connecting");
                headset.connect(CONNECT_METHOD_AUTO);
                   // perform your own code
    }
```

## 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.

The BlueParrott listener allows your application to listen for the stages associated with establishing a connection to the BlueParrott Button through the methods

- onConnect - when the SDK has successfully connected over bluetooth
- onConnectFailure  - when the SDK has failed to connect
- onConnectProgress - an event to monitor the steps in making the connection
- onValuesRead - after the sdk has connected it reads some headset values (firmware version etc.) and these are now available to read

### Connected

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

```
...
      //override method in IBPHeadsetListener
      @Override
      public void onConnect() {
          Log.d("Connected");
          //place your code here
      }
```

```
    @Override
    public void onValuesRead(){
        Log.d(TAG,"headset values are available to use");
        // can now read appname, firmware version etc.
    }
```

### Connection Failure

If the connection attempt fails the onConnectFailure method is called with an error Code which can be handled in your code.

```
//Method from class : com.blueparrott.blueparrottsdk.BPHeadsetListener
@Override
public void onConnectFailure(int errorCode) {
    //handle the connection failure here
    //this may include instruction "Retry or turn headset off then on"
}
```

| Reason Code | Value | Meaning |
|---|---|---|
| UPDATE_ANDROID | 1 | Android OS 4.4 or greater required for Classic Connection |
| BLUETOOTH_NOT_AVAILABLE | 2 | Bluetooth is not turned on |
| ALREADY_CONNECTED | 3 | Parrott Button is already connected |
| ALREADY_CONNECTING | 4 | Another connection attempt is already under way |
| NO_HEADSET_CONNECTED | 5 | There is no Bluetooth headset connected |
| HEADSET_NOT_SUPPORTED | 6 | Headset may not support BlueParrott Button |

| | | |
|---|---|---|
| UPDATE_YOUR_FIRMWARE | 7 | Firmware on the headset is not offering BlueParrott Button Service. Firmware may be too old |
| UPDATE_YOUR_SDK_APP | 8 | This SDK/App is too old to connect to the firmware version on headset |
| HEADSET_DISCONNECTED | 9 | Headset classic connection disconnected during BLE connection attempt |
| TIMEOUT | 10 | Unknown error, connection attempt has timed out |
| BLE_REQUIRES_PERMISSION | 11 | BLUETOOTH_SCAN and BLUETOOTH_CONNECT (Android 12) or BLUETOOTH_FINE_LOCATION (prior to Android 12) permission required for BLE Connection |
| CLASSIC_REQUIRES_PERMISSION | 12 | BLUETOOTH_CONNECT required for connection (>Android 12) |

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.

**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 class : com.blueparrott.blueparrottsdk.BPHeadsetListener
@Override
public void onConnectProgress(int progressCode) {
      Log.d(TAG, "Progress Code:"+progressCode);
      Log.d(TAG,getStatusDescription(progressCode));
}
```

| Status | Value | Meaning |
|---|---|---|

| | | |
|---|---|---|
| WAITING_TO_CONNECT | 0 | Connection attempt will commence shortly |
| STARTED | 1 | Connection attempt has started |
| FOUND_CLASSIC_HEADSET | 2 | A Bluetooth headset has been found |
| REUSING_CONNECTION | 3 | Another app is connected over BLE, attempting to reuse connection |
| BLE_SCANNING_ | 4 | Scanning for BLE services |
| FOUND_BP_SERVICE | 5 | The BlueParrott service has been found |
| CONNECTING_TO_BLE | 6 | Attempting to connect to the BlueParrott service over BLE |
| READING_HEADSET_VALUES | 7 | BLE connection established, reading settings from headset |
| USING_BT_CLASSIC | 8 | Connection being made using Bluetooth Classic |

## 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.

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

```
//enable headset SDK mode - enable your app to receive button events
```

```
if (!headset.sdkModeEnabled()) {
      logStatus("Enabling SDK...");
      headset.enableSDKMode();
}

//disable headsetSDKMode - put button back to mute mode
if (headset.sdkModeEnabled()) {
    headset.disableSDKMode();
    //do something...
}
```

## 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 |
|---|---|---|
| BUTTON_MODE_UNKNOWN | -1 | An unknown mode. This can be returned if the current mode has not been read yet. |
| BUTTON_MODE_MUTE | 0 | Mute on call mode. This is the default mode. |
| BUTTON_MODE_SPEEDDIAL | 1 | Speed dial a specific number. |
| BUTTON_MODE_PARTNER_APPLICATION | 2 | Compatible App or SDK Mode. |

## 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 class : com.blueparrott.blueparrottsdk.BPHeadsetListener
//Listen for success of SDK mode update
@Override
public void onModeUpdate() {
    logStatus("Mode Updated");
}
```

## Listen for BlueParrott Button Mode Update Failure

```
//listen for failure of mode update
@Override
public void onModeUpdateFailure (int reasonCode) {
      logStatus("Mode Update Failed. Reason"
+getUpdateErrorDescription(reasonCode));
      //handle error
}
```

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

| Reason Code | Value | Meaning |
|---|---|---|
| NOT_CONNECTED | 1 | BLE connection not available |
| WRITE_FAILED | 2 | Writing to headset over BLE failed |
| TIMEOUT | 3 | Operation timed out |

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 user's 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 class : com.blueparrott.blueparrottsdk.BPHeadsetListener
@Override
public void onButtonDown(int buttonId) {
    Log.d(TAG,"Button Down");
    //your code goes here
}

@Override
public void onButtonUp(int buttonId) {
    Log.d(TAG,"Button Up");
     //your code goes here
}

@Override
public void onTap(int buttonId) {
    Log.d(TAG,"Tap");
    //your code goes here
}

@Override
public void onDoubleTap(int buttonId) {
    Log.d(TAG,"DoubleTap");
    //your code goes here
```

```
@Override
public void onLongPress(int buttonId) {
    Log.d(TAG,"Long Press");
    //your code goes here
}
```

## Listening for Proximity Events

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

```
// Methods from protocol: BPHeadsetListener

@Override
public void onProximityChange(int status) {
   Log.d(TAG,"Proximity Change ="+status);

}
```

## 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.

```
    if (headset.connected()) {
                Log.d(TAG,"Disconnecting..");
                headset.disconnect();
     }
```

### 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.

```
public class BpSDKDemo extends AppCompatActivity implements
BPHeadsetListener {
      //add a BPHeadsetListener
    headset.addListener(this);

          @Override
          public void onDisconnect() {
              Log.d(TAG,"Disconnected");
               //tidy up here
          }
```

## 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.

## getAppKey

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).

## getAppName

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.

## getSpeedDialNumber

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()

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.

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:

## setAppMode

**Signature:**
setAppMode(String appKey, String appName, String partnerAppPackageName)

**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

**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

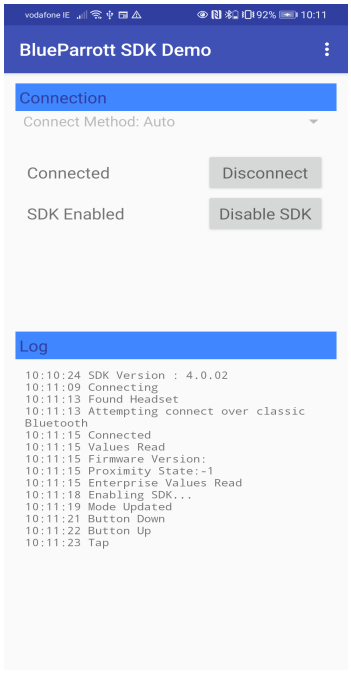
The Android SDK ‘BlueParrottSDKDemo’ Application is a sample of a simple integration, allowing the developer to get up and running quickly while using the BlueParrott SDK.

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. 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
- Utilizes BlueParrott listener recognising the BlueParrott Button clicks
- Recognises and logs Proximity events
- Disables BlueParrott SDK mode
- Disconnects from the headset BlueParrott Button
- Allows reading and setting of Enterprise config values

Below is a sample screen from the application:



# Additional Sample Applications

The SDK package also includes demo app ‘BlueParrottSDKConnectSample’ which gives an example template for how to maintain a connection to the headset SDK in a Service, in order to support background event handling.