

デベロッパーズガイド

Moverio Controller Function SDK

Seiko Epson Corporation

Trademarks

The product names, brand names, and company names mentioned in this guide are the trademarks or registered trademarks of their respective companies.

microSD and microSDHC are the trademarks or registered trademarks of the SD Card Association.

Wi-Fi®, Wi-Fi Direct™, and Miracast™ are the trademarks or registered trademarks of the Wi-Fi Alliance.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by the Seiko Epson Corporation is under license.

USB Type-C™ is a trademark of the USB Implementers Forum.

Google, Google Play, and Android are the trademarks of Google Inc.

Windows is the trademark or registered trademark of the Microsoft Corporation in the USA, Japan, and other countries.

Mac and Mac OS are the trademarks of Apple Inc.

Intel, Cherry Trail, and Atom are the trademarks of the Intel Corporation in the USA and other countries.

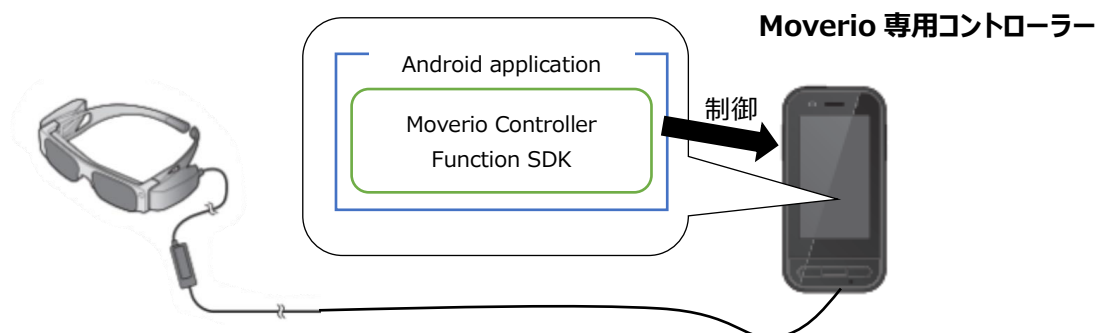
Other product names used herein are also for identification purposes only and may be trademarks of their respective owners. Epson disclaims any and all rights in those marks.

This material is not sponsored by Unity Technologies or its affiliates and is not affiliated with Unity Technologies or its affiliates. "Unity" is a trademark or registered trademark of Unity Technologies or its affiliates in the United States and other regions.

Moverio Controller Function SDK による Moverio 向けアプリ

開発の概要

Moverio Controller Function SDK を使用して BT-40 などの USB 接続型 Moverio と接続可能な専用コントローラー（型番：BO-IC400/BO-IC400N）のアプリケーションから、専用コントローラーのボタンの無効やキーコードのカスタマイズなどができます。



Moverio Controller Function SDK は、以下の開発環境をサポートします。

	Android Studio	Unity* 1
Android アプリケーション	●	●

*1 Unity 2018.4.0f1 以降をサポート

本ドキュメントでは Android Studio でのアプリ開発における Moverio Controller Function SDK で使用できる機能について説明します。

Unity を使用したアプリ開発については Moverio Controller Function SDK UnityPlugin デベロッパーズガイドを参照してください。

なおここで紹介する機能は Moverio 専用になります。一般の Android スマートフォンなどでは、機能を利用することはできませんのでご注意ください。

Moverio Controller Function SDK の適用範囲

Moverio Controller Function SDK は、専用コントローラーで動作する Android アプリ開発にのみ適用することができます。
Android スマートフォンや BT-300、BT-350 などの旧 Moverio 製品向けのアプリ開発には適用できないためご注意ください。

	BT-30ES (専用コントローラー+BT-30E)	BT-35ES (専用コントローラー+BT-35E)	BT-40S (専用コントローラー+BT-40)	BT-45CS (専用コントローラー+BT-45C)	一般の Android スマートフォン	旧 Moverio 製品 (BT-200/BT-2000/BT-2200/BT-300/BT-350)
Moverio Controller Function SDK V1.1.0	●	●	●	●	-	-

Moverio Controller Function SDK 機能一覧と専用コントローラーの OS バージョンの互換性

専用コントローラーの OS の各バージョンでサポートされている機能は以下になります。

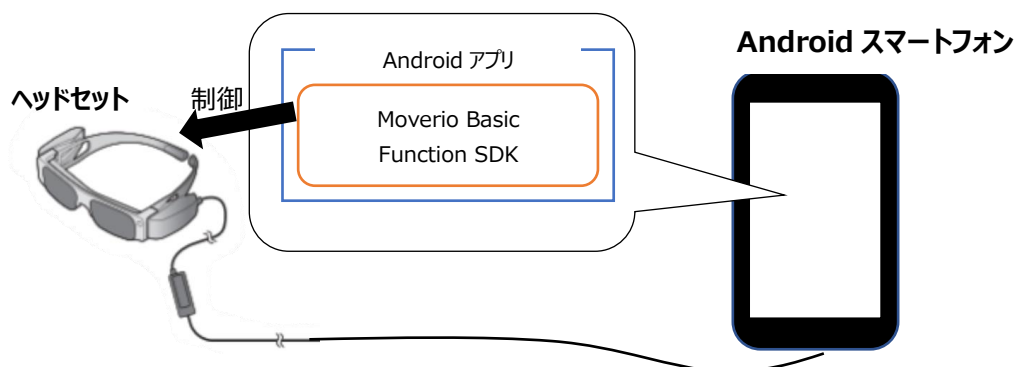
OS バージョン		HA24_R1.0.00/R1.0.01	HA24_R1.1.01	HA24_R2.0.00	HA24_R2.1.00
機能		HA24_E1.0.00/E1.0.01	HA24_E1.1.01	HA24_E2.0.00	HA24_E2.1.00
		HA24_V1.0.00/V1.0.01	HA24_V1.1.02	HA24_C2.0.00	HA24_C2.1.00
		HA24_C1.0.00/C1.0.01	HA24_C1.1.02	HA24_N2.0.00	HA24_N2.1.00
専用 API による制御	キー操作ロック	●	●	●	●
	キーコードカスタマイズ	●	●	●	●
	コントローラー LED 制御	●	●	●	●
	UI モード切替	-	-	●	● *2
Android CameraAPI によるカメラ制御		-	● *1	●	● *3
Android Sensor API によるセンサー制御		-	-	-	●

- *1 BT-35E カメラ制御のみサポート
- *2 十字キーによる画面操作ができる機能（Assist mode）を追加
- *3 Camera Custom State 設定の追加

専用コントローラーの OS が最新バージョンで無い場合は、一部機能を使用できません。そのためシステムアップデートによってコントローラーの OS を最新版にアップデートしてお使いください。

Moverio Basic Function SDK を使用したアプリ開発について

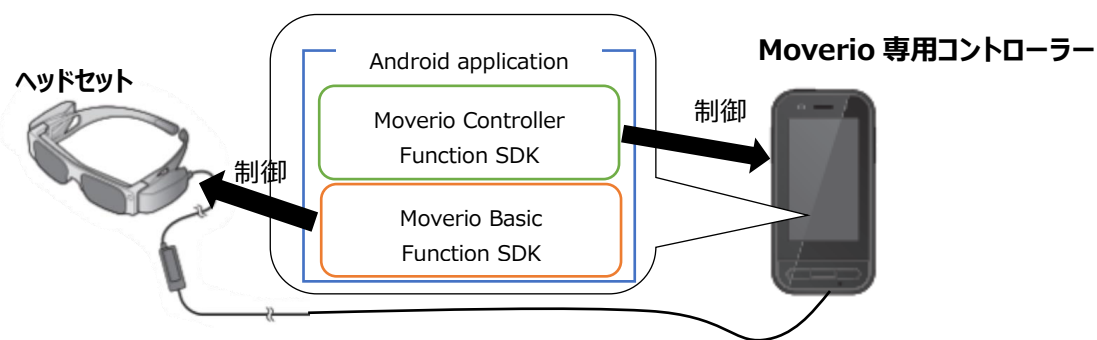
Moverio Basic Function SDK を使用すると表示設定や、モーションセンサーデータの取得、カメラのアクセスなど、ヘッドセットを制御できます。



Moverio Basic Function SDK には主に以下の機能があります。

- ・ヘッドセットディスプレイ制御機能
- ・ヘッドセットセンサー制御機能
- ・ヘッドセットカメラ制御機能
- ・ヘッドセットオーディオ制御機能

Moverio Basic Function SDK は専用コントローラーのアプリ開発にも使用することが可能で、Moverio Basic Function SDK をアプリに組み込むことで、専用コントローラーからヘッドセットのディスプレイの明るさの調整や 2D/3D 表示モードの切り替えなどを行うことができます。またアプリに Moverio Controller Function SDK を組み込むことで、専用コントローラーの制御も同一アプリから行うことができます。



Moverio Basic Function SDK の詳細は、Moverio Basic Function SDK のデベロッパーズガイドを参照ください。

Android アプリ開発の概要

専用コントローラー向けの Android アプリケーションを開発するために必要となる手順について記載します。

Android SDK の導入

Windows10 を搭載したパソコンに Android SDK を導入する方法について記載します。

Android Studio の入手

Android Studio を下記のサイトからダウンロードします。

<https://developer.android.com/studio/>

Android Studio のインストール

インストーラーの指示に従い Android Studio をインストールします。

例) C:\Users\<ユーザー名>\AppData\Local\Android\Sdk

※以降、上記フォルダに Android Studio がインストールされていることを前提に記載します。

Android Studio のプロキシ設定

プロキシ設定の必要なネットワーク環境でアプリケーションを開発する場合は、Android Studio のプロキシ設定を行ってください。下記のサイトで詳細な手順を確認してください。

<https://developer.android.com/studio/intro/studio-config#gradle-plugin>

プロキシ設定が不明な場合は、ネットワーク管理者にプロキシを利用した外部ネットワークへの接続方法についてお問い合わせください。

Android SDK Manager でのツールの取得・更新

アプリケーションの開発に必要なツール類は、Android SDK Manager を使用してください。下記のサイトで詳細な手順を確認してください。

https://developer.android.com/studio/intro/update#sdk_manager

ホスト機とパソコンの接続

ADB の接続確認コマンドでパソコンとホスト機器が接続されているか確認することができます。

コマンドプロンプトを起動し、"`cd C:\Users\<ユーザー名>\AppData\Local\Android\sdk\platform-tools`" を実行しフォルダを移動します。

※環境変数で上記の Path を通しておくくと便利です。

"adb devices"を実行してリストにデバイス名が表示されれば ADB 接続ができています。

```
c:\>cd Users\          ¥AppData¥Local¥Android¥sdk¥platform-tools

c:\Users¥!            ¥AppData¥Local¥Android¥sdk¥platform-tools>adb devices
List of devices attached
¥¥¥ device
```

※表示されない場合は、ホスト機器を USB 接続しなおし、再度"adb devices"を実行してください。

Moverio Controller Function SDK の組み込み

Moverio Controller Function SDK は、専用コントローラーを制御するための専用 API を有しています。専用 API の利用方法について説明します。

下記利用方法は、Android Studio でのアプリケーション開発を前提としています。

Android Studio の ProjectView を表示し、[File]--[New]--[Directory]で"libs"フォルダを作成します。

C:¥Users¥<ユーザー名>¥AndroidStudioProjects¥<アプリケーション名>¥app¥libs が作成されますので、ここへ Moverio Controller Function SDK_1.0.0.aar を置きます。

(作成したプロジェクトのフォルダが C:¥Users¥<ユーザー名>¥AndroidStudioProjects の場合)

※以降、C:¥Users¥<ユーザー名>¥AndroidStudioProjects¥<アプリケーション名>にプロジェクトがあるものとして説明しています。

その後、アプリケーションの最小 API レベルを 28 で定義し、アプリケーションの依存関係を追加してください。

```
apply plugin: 'com.android.application'

android {
    :
    // APIレベル28を追加
    minSdkVersion 28
    :
}

dependencies {
    :
    // Moverio Controller Function SDK.aarを追加
    implementation files('libs/Moverio Controller Function SDK_1.0.0.aar')
    :
    :
}
```

Android Studio 上部の Sync Project with Gradle Files ボタンを押して、Gradle の変更をプロジェクトに反映します。

ネットワークを介してアプリケーションをデバッグする

USB 接続型 Moverio のアプリケーション開発では、USB 接続型 Moverio と専用コントローラーを USB 接続してアプリケーション動作を確認します。このときパソコンと専用コントローラーは USB 接続されていないためデバッグできません。

上記を解決する方法として、ネットワークを介して専用コントローラーに adb 接続することで USB 接続型 Moverio と専用コントローラーを USB 接続したまま開発する方法について記載します。下記サイトも合わせて確認ください。

<https://developer.android.com/studio/command line/adb#wireless>

1. パソコンと専用コントローラーを同じネットワークに接続します。

2. パソコンと専用コントローラーを USB 接続します。

※以降、パソコンに接続されている専用コントローラーは 1 台のみとして記載します。

3. パソコンにおいて下記コマンドを実行し、専用コントローラーにネットワークを介して adb 接続できるようにします。

```
# adb tcpip 5555
```

4. パソコンと専用コントローラーの USB 接続を解除します。

5. 専用コントローラーの IP アドレスを確認します。

6. パソコンにおいて下記コマンドを実行し、専用コントローラーにネットワークを介して adb 接続します。

adb connect ip_address （例: adb connect 192.168.1.10#5555）

7. パソコンにおいて下記コマンドを実行し、パソコンに専用コントローラーが adb 接続されていることを確認します。

接続したデバイスが一覧に含まれることを確認します。

接続されていない場合は以下を確認してください。

- パソコンと Android スマートフォンが同じネットワークに接続されているか
- 3.からの手順を何度か実行し、接続できないか
- 6.で入力した IP アドレスに誤りはないか # adb devices

8. Android Studio を用いて専用コントローラーにアプリケーションをインストールし、アプリケーション動作を確認します。

Moverio Controller Function SDK 専用 API の説明

キー操作ロック

専用コントローラーのキー操作やタッチ操作をロックすることができます。操作ロックの対象は、電源ボタンや切替キーなどがあります。



専用コントローラーのキー操作やタッチ操作のロックには、com.epson.moverio.btcontrol.BtCustomKey をインポートしてください。

```
import com.epson.moverio.btcontrol.BtCustomKey;
```

BtCustomKey クラスのインスタンスを作成して、setKeyEnable()メソッドの第一引数に下表のキーの定数を指定し、第二引数にboolean 型のロック（false）、または、解除（true）を指定し、呼び出してください。

定数	説明
BtCustomKey.VOLUME_UP	音量アップキー
BtCustomKey.VOLUME_DOWN	音量ダウンキー
BtCustomKey.TRIGGER	切替キー
BtCustomKey.HOME	HOMEキー
BtCustomKey.BACK	BACKキー
BtCustomKey.APP_SWITCH	履歴キー
BtCustomKey.POWER	電源ボタン
BtCustomKey.ALLKEY	すべての物理キーを指定する際に使用する。

```

public class UIControlActivity extends Activity {
    private BtCustomKey mBtCustomKey = null;
    private Button mButton_powerKeyLock = null;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mBtCustomKey = new BtCustomKey(this);
        mButton_powerKeyLock = (Button) findViewById(R.id.button_powerKeyLock);
        mButton_powerKeyLock.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                mBtCustomKey.setKeyEnable(BtCustomKey.POWER, false); /* Disable power key. */
            }
        });
    }
    :
    :
}

```

キーコードのカスタマイズ

専用コントローラーの物理キーに割り当てられたキーコードを変更することができます。ただし、電源ボタンのキーコードを変更することはできません。

専用コントローラーのキーコードのカスタマイズには、com.epson.moverio.btcontrol.BtCustomKey をインポートしてください。

```
import com.epson.moverio.btcontrol.BtCustomKey;
```

BtCustomKey クラスのインスタンスを作成して、setKeyAssign()メソッドの第一引数に下表の対象の物理キーを指定し、第二引数に下表の変更可能なキーコードを指定し、呼び出してください。

対象の物理キー	変更可能なキーコード
BtCustomKey.VOLUME_UP	KeyEvent.KEYCODE_VOLUME_UP
BtCustomKey.VOLUME_DOWN	KeyEvent.KEYCODE_VOLUME_DOW
BtCustomKey.TRIGGER	KeyEvent.KEYCODE_FUNCTION
BtCustomKey.HOME	KeyEvent.KEYCODE_HOME
BtCustomKey.BACK	KeyEvent.KEYCODE_BACK
BtCustomKey.APP_SWITCH	KeyEvent.KEYCODE_APP_SWITCH
	KeyEvent.KEYCODE_F1
	KeyEvent.KEYCODE_F2
	KeyEvent.KEYCODE_F3

	KeyEvent.KEYCODE_F4
--	---------------------

```
public class UIControlActivity extends Activity {
    private BtCustomKey mBtCustomKey = null;
    private Button mButton_appSwitch2Home = null;

    @Override public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mBtCustomKey = new BtCustomKey(this);
        mButton_appSwitch2Home = (Button) findViewById(R.id.button_appSwitch2Home);
        mButton_appSwitch2Home.setOnClickListener(new View.OnClickListener() {

            @Override public void onClick(View view) {
                mBtCustomKey.setKeyAssign(BtCustomKey.APP_SWITCH, KeyEvent.KEYCODE_HOME);
            }
        });
    }
    :
    :
}
```

キー状態の初期化

キー操作ロック、及びキーコードカスタマイズにより変更したキー状態を初期状態に戻すことができます。

キー操作ロック、またはキーコードカスタマイズで生成したインスタンスで、resetToDefault()を実行してください。

```
mBtCustomKey.resetToDefault();
```

※キー操作によるキー状態の初期化：“切替キー、ホームキー、音量 Up キーを同時に 3 秒間押し続ける”ことによって、キー状態を初期化することができます。キー状態を変更したままアプリが異常終了してしまった場合に強制的にキー状態を元に戻す場合に使用して下さい。

コントローラーLED 制御

専用コントローラーの LED インジケーターの点灯/非点灯を制御することができます。



コントローラーLED の制御には、com.epson.moverio.btcontrol.BtControllerLedMode をインポートしてください。

```
import com.epson.moverio.btcontrol.BtControllerLedMode;
```

BtControllerLedMode クラスのインスタンスを作成して、setControllerLedMode()メソッドの第一引数に下表のキーの定数を指定して呼び出してください。

定数	説明
BtControllerLedMode.MODE_OFF	LEDを非点灯モードにする
BtControllerLedMode.LED_MODE_NORMAL	LEDを点灯モードにする

```
public class UIControlActivity extends Activity {  
    private BtControllerLedMode mBtLed = null;  
    private Button mButton_ledOff = null;  
  
    @Override public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mBtLed = new BtControllerLedMode (this);  
        mButton_ledOff = (Button) findViewById(R.id.button_appSwitch2Home);  
        mButton_ledOff.setOnClickListener(new View.OnClickListener() {  
            @Override public void onClick(View view) {  
                mBtLed.setControllerLedMode(BtControllerLedMode.MODE_OFF);  
            }  
        });  
    }  
    :  
    :  
}
```

UI モード切替

専用コントローラーにはヘッドセットとコントローラーに同じ映像が表示されるミラーモードと、コントローラーをトラックパッドとして使用することができるトラックパッドモード、コントローラーを十字キーとして使用することができるアシストモードがあります。

これらのモードを本 API 切り替えることができます。

UI モード切替には、com.epson.moverio.btcontrol.UIControl をインポートしてください。

```
import com.epson.moverio.btcontrol.UIControl;
```

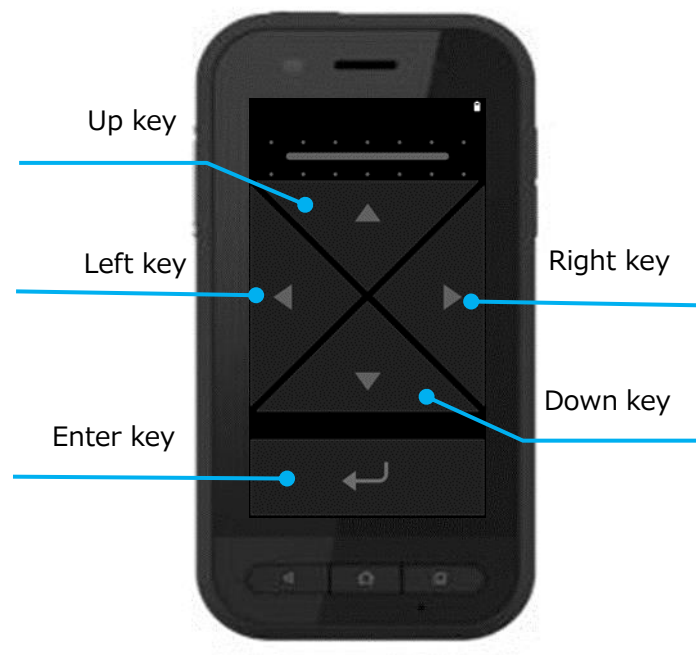
UIControl クラスのインスタンスを作成して、setUiMode()メソッドの第一引数に下表のキーの定数を指定して呼び出してください。

定数	説明
UIControl.UI_MODE_MIRROR	ミラーモードに切り替える
UIControl.UI_MODE_TRACK	トラックパッドモードに切り替える
UIControl.UI_MODE_ASSIST	アシストモードに切り替える

```
public class UIControlActivity extends Activity {  
    private UIControl mUiMode = null;  
    private Button mButton_modeTrack = null;  
  
    @Override public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mUiMode = new UIControl (this);  
        mButton_modeTrack = (Button) findViewById(R.id.button_appSwitch2Home);  
        mButton_modeTrack.setOnClickListener(new View.OnClickListener() {  
            @Override public void onClick(View view) {  
                mUiMode.setUiMode(UIControl. UI_MODE_TRACK);  
            }  
        });  
    }  
    :  
    :  
}
```

※本 API でモードをミラーモードからトラックパッドモード、またはアシストモードへ切り替える時、アプリ画面を再構築するために一度アプリが自動的に終了します。モード切替が完了した後、終了する前のアプリ画面（アクティビティ）が自動的に再表示されます。

専用コントローラーの OS バージョン 2.1.00 より、コントローラーを十字キーとして使用することができるアシストモードが追加されました。アシストモードに切り替わるとコントローラーに十字キーとエンターキーが表示され、キー操作に応じてアプリは Android API の dispatchKeyEvent()により以下のキーイベントを取得することができます。



キーの名称	キーイベント
Up key	KeyEvent.KEYCODE_DPAD_UP
Down key	KeyEvent.KEYCODE_DPAD_DOWN
Left key	KeyEvent.KEYCODE_DPAD_LEFT
Right key	KeyEvent.KEYCODE_DPAD_RIGHT
Enter key	KeyEvent.KEYCODE_DPAD_CENTER

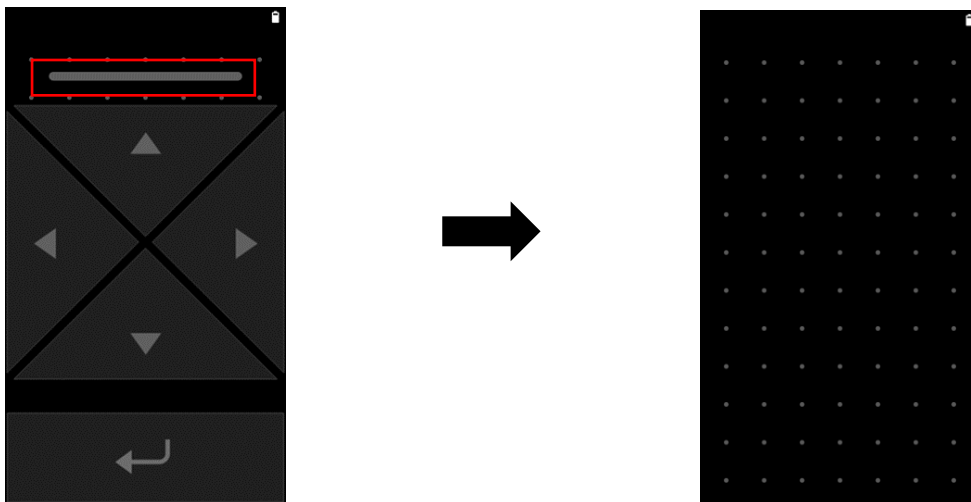
```

@Override
public boolean dispatchKeyEvent(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.KEYCODE_DPAD_UP && e.getAction() == KeyEvent.ACTION_DOWN) {
        // Press Up key
    } else if(e.getKeyCode() == KeyEvent.KEYCODE_DPAD_DOWN && e.getAction() ==
KeyEvent.ACTION_DOWN) {
        // Press Down key
    } else if(e.getKeyCode() == KeyEvent.KEYCODE_DPAD_LEFT && e.getAction() ==
KeyEvent.ACTION_DOWN) {
        // Press Left key
    } else if(e.getKeyCode() == KeyEvent.KEYCODE_DPAD_RIGHT && e.getAction() ==
KeyEvent.ACTION_DOWN) {
        // Press Right key
    } else if(e.getKeyCode() == KeyEvent.KEYCODE_DPAD_CENTER && e.getAction() ==
KeyEvent.ACTION_DOWN) {
        // Press Enter key
    }

    return true;
}

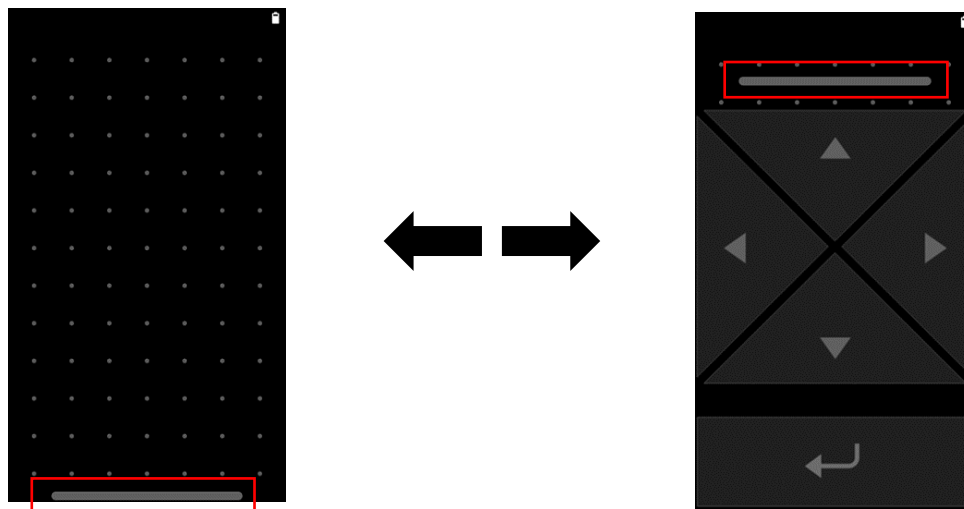
```

十字キーの上部に表示されているバーを引き下げると、アシストモードは終了し、トラックパッドモードになります。



設定->ユーザー補助->Arrow Key のトグルボタンを ON にすると、setUiMode()を使用しなくてもアシストモードが使用できるようになります。（デフォルト設定は OFF になります）

上記設定を ON にするとトラックパッドモードの下側にバーが表示されます。Trackpad mode でこのバーを引き上げるとアシストモードになり、Assist mode でこのバーを引き下げるとトラックパッドモードに戻ります。



※アシストモードは新規アプリケーション開発向けの機能になります。プリインアプリも含めた既存のアプリは基本的に十字キー/エンターキーによる操作に対応していない（画面操作できない）ためご注意ください。

キーロック機能の切替

専用コントローラーの切替キーを長押した時に実行されるキーロック機能の有効/無効を切り替えることができます。

キーロックを切り替えるには、com.epson.moverio.btcontrol.KeyLock をインポートしてください。

```
import com.epson.moverio.btcontrol.KeyLock;
```

KeyLock クラスのインスタンスを作成して、setKeyLock()メソッドの第一引数に下表のキーの定数を指定し、呼び出してください。

定数	説明
KeyLock.KEY_LOCK_ENABLE	キーロック機能有効
KeyLock.KEY_LOCK_DISABLE	キーロック機能無効

※キーロック機能が有効になっている場合、全ての専用コントローラーの操作（キー操作、及びタッチ操作）が無効になります。キーロック機能を有効にする必要が無い場合は、速やかに無効にしてください。以下のコードはキーロック機能を有効にした 10 秒後に自動でキーロック機能を無効にしています。また専用コントローラーがスリープ状態に入る時にキーロックは自動で無効になります。

```

public class UIControlActivity extends Activity {
    private KeyLock mKeyLock = null;
    private Button mButton_keyLock = null;

    @Override public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mKeyLock = new KeyLock (this);
        mButton_keyLock = (Button) findViewById(R.id.button_keyLock);
        mButton_keyLock.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                mKeyLock.setKeyLock(KeyLock.KEY_LOCK_ENABLE);
                Handler handler = new Handler(getMainLooper());
                handler.postDelayed(new Runnable(){
                    public void run() {
                        if (keyLock.getKeyLock() == KeyLock.KEY_LOCK_ENABLE) {
                            keyLock.setKeyLock(KeyLock.KEY_LOCK_DISABLE);
                        }
                    }
                }, 10000);
            }
        });
    }
    :
    :
}

```

Android CameraAPI によるカメラ制御

BT-45C や BT-35E の USB 接続型 Moverio は装着者の前面を撮影するカメラをヘッドセットに搭載しており、カメラの映像データの取得や静止画/動画の撮影などができます。カメラの映像データを使用することで、通常の撮影に加え、マーカ認識などの用途に応用することができます。

ここでは専用コントローラー を用いて BT-45C、BT-35E カメラ制御する方法について説明します。

*なお本章での説明は、専用コントローラーの OS バージョン 2.0.00、2.1.00 の内容になります。

専用コントローラーを用いた BT-45C/BT-35E カメラ制御方法

Android CameraAPI や Moverio Basic Function SDK を使用すると、専用コントローラー を用いて BT-45C/BT-35E に搭載されているカメラを制御することができます。Moverio Basic Function SDK による制御方法についてはデベロッパーズガイドを参照してください。

ただし、Android CameraAPI と Moverio Basic Function SDK によるカメラ制御は両立することができません。Moverio Basic Function SDK によるカメラ制御を行った後は、Android CameraAPI からはカメラを検出することができなくなります。Moverio Basic Function SDK によるカメラ制御使用後に Android CameraAPI を使用する場合は、BT-45C/BT-35E を挿抜してください。

Android CameraAPI は Android CameraAPI1 と Android CameraAPI2 があります。各 API の使用方法は以下の情報を参照してください。

✓ Android CameraAPI1(android.hardware.Camera)

<https://developer.android.com/training/camera>

※CameraAPI1 は非推奨です

✓ Android CameraAPI2(android.hardware.camera2)

<https://developer.android.com/training/camera2>

<https://medium.com/google-developers/detecting-camera-features-with-camera2-61675bb7d1bf>

✓ Android CameraAPI 2 サンプルソースコード

以下は Android CameraAPI2 を使用したカメラアプリのサンプルソースコードです。アプリをビルドし、BT-45C/BT-35E を接続してアプリを起動すると、BT-45C/BT-35E カメラを使用することができます。アプリ実装の参考にしてください。

<https://github.com/googlearchive/android-Camera2Basic>

Android API/Moverio Basic Function SDK のカメラ制御可能項目

専用コントローラー を用いて BT-45C/BT-35E に搭載されているカメラを Android CameraAPI2 により制御することができます。ただし、一部の設定は Android CameraAPI2 では変更できません。変更するためには Moverio Basic Function SDK を使用してください。BT-45C/BT-35E のカメラスペック、及び各 API/SDK による設定の制御可否を“Android CameraAPI2/Moverio Basic Function SDK でのカメラ制御可能対応表”に記載しています。

Android CameraAPI2/Moverio Basic Function SDK でのカメラ制御可能対応表

Settings	BT-35E	BT-45C	Android	Moverio
----------	--------	--------	---------	---------

			CameraAPI	Basic Function SDK
解像度	640x480	640x480	●	●
	1280x720	1280x720	●	●
		1600×1200	●	●
	1920x1080	1920x1080	●	●
		2560×1440	●	●
	2592x1944		●	●
		3264×2448	●	●
露出補正モード	auto/manual	auto/manual	auto only	●
手動露出補正ス テップ	-5 ~ +5	-7 ~ +5		●
フォーカスモード	Off (Pan)	auto/manual	auto only	●
手動フォーカス		+50 ~ +3000[mm]		●
シャープネス	0 ~ +128			●
明るさ	-127 ~ +127	0 ~ +255		●
ゲイン	0 ~ +255	+1024 ~ +16383		●
ホワイトバランス	auto/cloudy_daylight(600 0K)/daylght(5500K)/fluore scent(4200K)/incandescen t(3200K)/twilight(3500K)	auto/cloudy_daylight(60 00K)/ daylight(5500K)/fluoresc ent(4200K)/ incandescent(3200K)/twi light(3500K)	auto only	●
電力線周波数	50Hz/60Hz	50Hz/60Hz/Disable		●
インジケータモー ド	auto	Auto/on/off		●

ヘッドセットカメラの優先使用機能について

専用コントローラーにはカメラ（バックカメラ）が搭載されており、通常 Android CameraAPI は専用コントローラーバックカメラを使用するようになっています。それに対して専用コントローラーには BT-45C/BT-35E カメラ（以下、ヘッドセットカメラ）を優先して使用できるようにする機能があります（以下、ヘッドセットカメラ優先使用機能）。機能のソフトウェア仕様については” Android CameraAPI を使用したカメラ制御仕様”を参照してください。ただし、OS バージョン 2.1.00 よりヘッドセットカメラ優先使用機能を利用するか選択できる Camera Custom State の設定が追加されています。

ヘッドセットカメラ優先使用機能は、OS バージョン 2.1.00 では以下の条件において有効になります。

- ✓ Camera Custom State が ON
- ✓ 専用コントローラーにヘッドセットカメラ（BT-45C、または BT-35E）が接続されている

✓ 専用コントローラーのモードがトラックパッドモード、またはアシストモードである

OS バージョン 2.0.00 でもヘッドセットカメラ優先使用機能は搭載されていますが、有効条件がバージョン 2.1.00 と一部異なりますのでご注意ください。“ヘッドセットカメラ優先使用機能 有効条件”で、OS バージョン 2.0.00 と 2.1.00 における条件をまとめています。なお、表における“-”は、通常の Android CameraAPI による制御状態にあることを示します。

ヘッドセットカメラ優先使用機能 有効条件							
専用コントローラーのモード	OS バージョン	バージョン 2.0.00		バージョン 2.1.00			
	その他 条件	Camera Custom State は無し		Camera Custom State がオン		Camera Custom State がオフ	
		H カメラ*接続	H カメラ非接続	H カメラ接続	H カメラ非接続	H カメラ接続	H カメラ非接続
ミラーモード		●	-	-	-	-	-
トラックパッドモード アシストモード		●	-	●	-	-	-

*H カメラ：ヘッドセットカメラ

Camera Custom State はデフォルトでオフとなっていますが、BT-45C、または BT-35E が初めて接続された時に自動でオンになります。設定を変更したい場合は、設定->ユーザー補助-> Camera Custom State のトグルボタンから設定を変更してください。二回目以降の接続時に自動でオンになることはありません。

Android CameraAPI を使用したカメラ制御仕様

ヘッドセットカメラ優先使用機能利用時は Android CameraAPI の一部仕様が通常の Android CameraAPI から異なっています。以下では、CameraAPI2 を例に、ヘッドセットカメラ優先使用機能の制御仕様について記載します。

カメラ特徴を取得するコードスニペット

```
CameraManager manager = (CameraManager) getSystemService(CAMERA_SERVICE);
try {
    for (String cameraId : manager.getCameraIdList()) {
        CameraCharacteristics chars = manager.getCameraCharacteristics(cameraId);
        // Do something with the characteristics
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }
}
```

“カメラ特徴を取得するコードスニペット”は cameraId を指定し、カメラ特徴を取得します。専用コントローラー は USB ポートが二つあり（ボトムポート、サイドポート）、ヘッドセットカメラや市販の USB カメラを接続することでカメラ制御を行うことができます。ただし、ヘッドセットカメラはボトムポートの接続のみサポートしています。

ヘッドセットカメラを専用コントローラーのボトムポートに接続しヘッドセットカメラ優先使用機能を利用している場合、“ヘッドセットカメラ接続時の cameraId”のように cameraId が割り当てられます。

ヘッドセットカメラ接続時の cameraId

Camera API	専用コントローラーバックカメラ	ボトムポート	サイドポート *
Camera API1	1	0	2
Camera API2	/dev/video3	0	/dev/video4

* USB カメラをサイドポートに接続している場合

CameraCharacteristics を参照することでカメラ特徴を取得し、所望するカメラを選択することができます。

カメラレンズ属性を取得するコードスニペット

```
CameraCharacteristics chars = manager.getCameraCharacteristics(cameraId);
Integer facing = chars.get(CameraCharacteristics.LENS_FACING);
```

“カメラレンズ属性を取得するコードスニペット”は cameraId を指定し、取得したカメラ特徴からカメラのレンズ属性を取得します。ヘッドセットカメラを専用コントローラーに接続しヘッドセットカメラ優先使用機能を利用している場合、レンズ属性は“ヘッドセットカメラ接続時のレンズ属性”のようになります。

ヘッドセットカメラ接続時のレンズ属性

Camera API	専用コントローラーバックカメラ	ボトムポート	サイドポート *
Camera API1	FRONT	BACK	FRONT
Camera API2	EXTERNAL	BACK	EXTERNAL

* USB カメラをサイドポートに接続している場合

“ヘッドセットカメラを使用するコードスニペット”は、ヘッドセットカメラを使用する場合のコードスニペットです。

ヘッドセットカメラを使用するコードスニペット

```
CameraManager manager = (CameraManager) getSystemService(CAMERA_SERVICE);
try {
    for (String cameraId : manager.getCameraIdList()) {
        CameraCharacteristics chars = manager.getCameraCharacteristics(cameraId);
        Integer facing = chars.get(CameraCharacteristics.LENS_FACING);
        if (facing != null && facing == CameraCharacteristics.LENS_FACING_BACK) {
            // Open BT-45C/BT-35E camera (Please refer to upper link)
        }
    }
} catch (CameraAccessException e) {
    e.printStackTrace();
}
```

cameraId の情報は、カメラを使用する他に、カメラのフラッシュライトを使用する API である setTorchMode でも利用します。“ヘッドセットカメラ接続時の cameraId”は、setTorchMode についても適用されますので、ヘッドセットカメラを接続しヘッドセットカメラ優先使

用機能を利用している状態で専用コントローラーの背面フラッシュライトを利用したい場合は、cameraId として"/dev/video3"を指定する必要があります。

■ USB カメラの使用について

専用コントローラー に USB カメラを接続する場合、ヘッドセットカメラ優先使用機能の状態に関わらず cameraId は"USB カメラ接続時の cameraId"のように割り当てられます。

USB カメラ接続時の cameraId

Camera API	専用コントローラーバックカメラ	ボトムポート *	サイドポート *
Camera API1	0	1 or 2	1 or 2
Camera API2	0	/dev/video3 or 4	/dev/video3 or 4

* USB ポートは先に接続された USB カメラに小さい番号が割り当てられます

レンズ属性は"USB カメラ接続時のレンズ属性"のようになります。

USB カメラ接続時のレンズ属性

Camera API	専用コントローラーバックカメラ	ボトムポート *	サイドポート *
Camera API1	BACK	FRONT	FRONT
Camera API2	BACK	EXTERNAL	EXTERNAL

USB カメラを専用コントローラーで使用する場合は、"USB カメラ接続時の cameraId"、及び"USB カメラ接続時のレンズ属性"のパラメータを、"ヘッドセットカメラを使用するコードスニペット"のソースコード内で指定してください。

■ 画面回転処理について

Android CameraAPI 2 サンプルソースコードでは、コントローラカメラの向きに応じた適切なカメラ表示を行うために、コントローラーの回転状態に応じた画面回転処理を行っています。

しかし、ヘッドセットカメラを用いて同処理を行う場合、コントローラーの回転状態とヘッドセットカメラの回転状態は連動しないため、表示に不整合が生じます。そのため、ヘッドセットカメラを使用する場合は"回転処理を含めないソースコード"のように回転処理を含めない (matrix.postRotate()をコメントアウトする) ことを推奨します。

回転処理を含めないソースコード

```
if (Surface.ROTATION_90 == rotation || Surface.ROTATION_270 == rotation) {  
    bufferRect.offset(centerX - bufferRect.centerX(), centerY - bufferRect.centerY());  
    matrix.setRectToRect(viewRect, bufferRect, Matrix.ScaleToFit.FILL);  
    float scale = Math.max((float) viewHeight / mPreviewSize.getHeight(), (float) viewWidth /  
mPreviewSize.getWidth());  
    matrix.postScale(scale, scale, centerX, centerY);  
    // matrix.postRotate(90 * (rotation - 2), centerX, centerY);  
}
```

また、専用コントローラーに搭載されているトラックパッドモード、アシストモードにおいては、"画面回転に関する AndroidAPI"のように画

面回転に関する AndroidAPI の返り値が固定値となっていますのでご注意ください。

画面回転に関する AndroidAPI	
Android API	返り値
Display#getRotation()	ROTATION_0
Configuration#orientation	ORIENTATION_LANDSCAPE

■ プレビューのアスペクト比について

専用コントローラー に搭載されているバックカメラとヘッドセットカメラでは取り付け角度が異なります。詳しくは、下記表を参照ください。

CameraCharacteristics.get(CameraCharacteristics.SENSOR_ORIENTATION)の返り値

カメラ種別	返り値
専用コントローラー バックカメラ	90
ヘッドセットカメラ	0

そのため、ヘッドセットカメラを使用する場合に取り付け角90 度を前提としたプレビューを行うと、プレビュー画面が歪む場合があります。（例えば、正方形の撮影物が長方形として表示されます）取り付け角度に注意し、取得される画像の解像度に適切なプレビューを行うことを推奨します。

Android CameraAPI 2 サンプルソースコードでは取り付け角90 度を前提としたプレビューを行っています。“適切なアスペクト比に変更するコード”のような修正をすることでヘッドセットカメラで適切なアスペクト比によるプレビューを行うことができます。

適切なアスペクト比に変更するコード

```
if(mSensorOrientation != 0){
    // When the mounting angle is not 0-degree, it is dedicated controller back camera, so it should be the
    // same as the original source code.
    if (orientation == Configuration.ORIENTATION_LANDSCAPE) {
        mTextureView.setAspectRatio(mPreviewSize.getWidth(),mPreviewSize.getHeight());
    } else {
        mTextureView.setAspectRatio(
            mPreviewSize.getHeight(), mPreviewSize.getWidth());
    }
} else {
    // When the mounting angle is 0-degree, it is determined to be BT-35E camera (USB camera).
    //BT-35E camera doesn't rotate regardless of the device state, so the aspect ratio is always assumed to
    // be horizontal preview.
    mTextureView.setAspectRatio(
        mPreviewSize.getWidth(), mPreviewSize.getHeight());
}
```



```
if(mSensorOrientation !=0) {
```

```
    // For BT-35E camera, the following process is not necessary because BT-35E camera doesn't rotate  
    regardless of the device state.
```

```
    if (Surface.ROTATION_90 == rotation || Surface.ROTATION_270 == rotation) {  
        bufferRect.offset(centerX - bufferRect.centerX(), centerY - bufferRect.centerY());  
        matrix.setRectToRect(viewRect, bufferRect, Matrix.ScaleToFit.FILL);  
        float scale = Math.max((float) viewHeight / mPreviewSize.getHeight(), (float) viewWidth /  
mPreviewSize.getWidth());  
        matrix.postScale(scale, scale, centerX, centerY);  
        matrix.postRotate(90 * (rotation - 2), centerX, centerY);  
    } else if (Surface.ROTATION_180 == rotation) {  
        matrix.postRotate(180, centerX, centerY);  
    }  
}
```

Android Sensor API によるセンサー制御

BT-45C、BT-40、BT-35E の USB 接続型 Moverio は動き、向き、環境を検知するセンサー（以下、ヘッドセットセンサー）を搭載しています。これにより装着者の頭の動きを推測したり、周辺環境の明るさを推測したりできます。

ここでは専用コントローラーを用いてヘッドセットセンサーを制御する方法について説明します。

*なお本章での説明は、専用コントローラーの OS バージョン 2.1.00 の内容になります。

専用コントローラーを用いたセンサー制御方法

Android Sensor API や Moverio Basic Function SDK を使用すると、専用コントローラーを用いてヘッドセットセンサーを制御することができます。Moverio Basic Function SDK による制御方法についてはデベロッパーズガイドを参照してください。

ただし、Android Sensor API と Moverio Basic Function SDK によるセンサー制御は両立することができません。Moverio Basic Function SDK によるセンサー制御を行った後は、Android Sensor API からヘッドセットセンサーを検出することができなくなります。Moverio Basic Function SDK によるセンサー制御使用後に Android Sensor API を使用する場合は、ヘッドセットを挿抜してください。

Android Sensor API の仕様については以下の情報を参照してください。

<https://developer.android.com/reference/android/hardware/Sensor>

ヘッドセットセンサーの優先使用機能について

専用コントローラーにはヘッドセットセンサーと同じ種類のセンサーが搭載されており、通常 Android Sensor API はコントローラー搭載のセンサーを使用するようになっています。それに対して専用コントローラーにはヘッドセットセンサーを優先して使用できる機能があります（以下、ヘッドセットセンサー優先使用機能）。またヘッドセットセンサー優先使用機能を利用するか選択できる Sensor Custom State の設定もあります。

ヘッドセットセンサー優先機能は、OS バージョン 2.1.00 では以下の条件において有効になります。

- ✓ Sensor Custom State が ON
- ✓ 専用コントローラーにヘッドセットセンサー（BT-45C、BT-40、または BT-35E）が接続されている
- ✓ 専用コントローラーのモードがトラックパッドモード、またはアシストモードである

“ヘッドセットセンサー優先使用機能 有効条件”で、ヘッドセットセンサー優先機能の挙動についてまとめています。なお、表における“-”は、通常の Sensor Custom API による制御状態にあることを示します。

ヘッドセットセンサー優先使用機能 有効条件

専用コントローラーのモード	Sensor Custom State が ON		Sensor Custom State が OFF	
	H センサー*接続	H センサー*非接続	H センサー接続	H センサー非接続
ミラーモード	-	-	-	-
トラックパッドモード アシストモード	●	-	-	-

*H センサー：ヘッドセットセンサー

Sensor Custom State はデフォルトでオフとなっていますが、BT-45C、BT-40、または BT-35E が初めて接続された時のみ自動でオンになります。設定を変更したい場合は、設定->ユーザー補助-> Sensor Custom State のトグルボタンから設定を変更してください。

い。二回目以降の接続時に自動でオンになることはありません。

ヘッドセットがサポートしているセンサータイプとセンサー軸について

ヘッドセットには様々な種類のセンサーが搭載されています。各ヘッドセットでサポートしている（データを利用できる）センサータイプを“ヘッドセットでサポートしているセンサータイプ”に記載しています。

ヘッドセットでサポートしているセンサータイプ

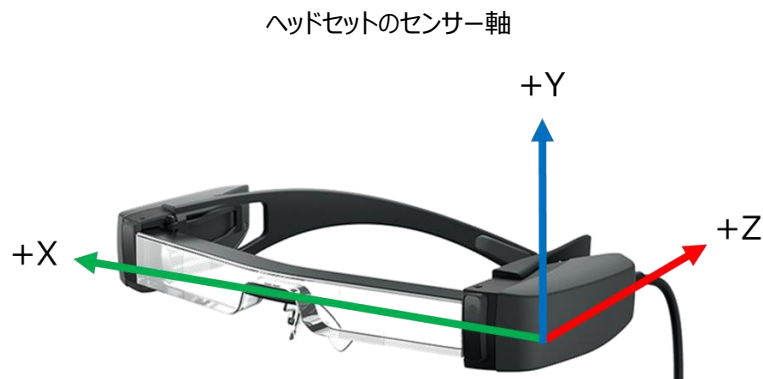
センサータイプ	センサーID (Hex)	説明	Android 標準 センサータイプ	イベント 周期	サポートしているセンサータイプ	
					BT-45C BT-40	BT-35E
TYPE_ACCELEROMETER	0x00000001	重力を含むヘッドセットの加速度を 3 軸（x、y、z）の加速度[m/s ²]で測定します。	●	100Hz	●	●
TYPE_MAGNETIC_FIELD	0x00000002	ヘッドセット周囲の地磁気を 3 軸（x、y、z）の地磁気[μT]で測定します。	●	100Hz	●	●
TYPE_GYROSCOPE	0x00000004	ヘッドセットの角速度を 3 軸（x、y、z）の角速度[rad/s]で測定します。	●	100Hz	●	●
TYPE_LIGHT	0x00000005	ヘッドセット周囲の照度[lx]を測定します。	●	5Hz	●	●
TYPE_GRAVITY	0x00000009	重力を 3 軸（x、y、z）のヘッドセットの加速度[m/s ²]で測定します。	●	100Hz	●	●
TYPE_LINEAR_ACCELERATION	0x0000000a	重力を除いた 3 軸（x、y、z）のヘッドセットの加速度[m/s ²]で測定します。	●	100Hz	●	●
TYPE_ROTATION_VECTOR	0x0000000b	ヘッドセットの向きを回転ベクトルで測定します。	●	100Hz	●	●
TYPE_MAGNETIC_FIELD_UNCALIBRATED	0x0000000e	ヘッドセット周囲の地磁気を未校正の 3 軸（x、y、z）で測定します。	●	100Hz	●	
TYPE_GAME_ROTATION_VECTOR	0x0000000f	ヘッドセットの向きを、地磁気を使用しない回転ベクトルで測定します。	●	100Hz	●	
TYPE_GYROSCOPE_UNCALIBRATED	0x00000010	ヘッドセットの角速度を未校正の 3 軸（x、y、z）	●	100Hz	●	

		の角速度[rad/s]で測定します。				
TYPE_STATIO NARY_DETECT	0x0000001d	ヘッドセットの静止を検知します。ヘッドセットが静止して 1 分経過すると"1"のデータが出力されます。	●	イベント 発生時	●	
TYPE_MOTION _DETECT	0x0000001e	ヘッドセットの動作を検知します。ヘッドセットの静止を検知した状態からヘッドセットを動かすと"1"のデータが出力されます。	●	イベント 発生時	●	
TYPE_ACCELE ROMETER_UN CALIBRATED	0x00000023	重力を含むヘッドセットの加速度を未校正の 3 軸 (x、y、z) の加速度 [m/s ²]で測定します。	●	100Hz	●	
TYPE_HEADSE T_TAP	0x00102001	ヘッドセットのタップを検知します。ヘッドセットがダブルタップされると"1"のデータが出力されます。	-	イベント 発生時	●	

ヘッドセットセンサー優先使用機能が有効状態で、アプリは任意のセンサーID を Android Sensor API に指定することでヘッドセットセンサーのデータを取得できます。

TYPE_HEADSET_TAP 以外のセンサータイプは Android 標準センサータイプになるため、Sensor クラスの定数をセンサーID として使用できます。TYPE_HEADSET_TAP は Android 標準センサータイプでないため、“ヘッドセットでサポートしているセンサータイプ”のセンサーID (0x00102001) を直接指定してください。

ヘッドセットセンサーのセンサー軸は、Android 標準のセンサーの座標系と同じです。ヘッドセットを装着した場合、X 軸は右方向を指し、Y 軸は上方向を指し、Z 軸は装着者の方向を指します。



なおヘッドセットセンサー優先使用機能が有効状態でヘッドセットセンサーを使用する場合、以下の点に御留意ください。

- ✓ ヘッドセットセンサーから出力されるデータのイベント周期は、“ヘッドセットでサポートしているセンサータイプ”のイベント周期に記載されている内容で固定されています。センサーを登録する際にイベント周期を変更しようとしても (registerListener() で SENSOR_DELAY_FASTEST を指定する、など) 反映されません。

- ✓ Sensor クラスで取得できる各センサーのプロパティ情報（名称、データ取得範囲、など）はヘッドセットセンサーのものでなく、元のセンサータイプであるコントローラーセンサーのプロパティ情報になります。
- ✓ BT-35E で非サポートのセンサータイプ（TYPE_GAME_ROTATION_VECTOR、TYPE_ACCELEROMETER_UNCALIBRATED、など）は Android 標準センサータイプのセンサーID を指定しても使用できません（センサーデータが出力されません）。コントローラーのセンサーを使用したい場合は、“ヘッドセットセンサー優先使用機能の有効状態におけるコントローラーセンサーデータの取得について”を参照してください。

ヘッドセットセンサー優先使用機能の有効状態におけるコントローラーセンサーデータの取得について

ヘッドセットセンサー優先使用機能が有効の状態では、センサーID を Android Sensor API に指定してもコントローラーからセンサーを使用することができません。ヘッドセットセンサー優先使用機能が有効の状態ではコントローラーセンサーのデータを使用したい場合は、専用コントローラー用カスタムセンサータイプのセンサーID を Android Sensor API に指定してください。

カスタムセンサータイプのセンサーID は、“専用コントローラー用カスタムセンサータイプ”に記載しています。なおこれらのセンサータイプは Android 標準センサータイプでないため、表に記載しているセンサーID を Android Sensor API に直接指定してください。

専用コントローラー用カスタムセンサータイプ

カスタムセンサータイプ	センサーID (Hex)	対応する Android 標準センサータイプ
TYPE_CONTROLLER_ACCELEROMETER	0x00100001	TYPE_ACCELEROMETER
TYPE_CONTROLLER_MAGNETIC_FIELD	0x00100002	TYPE_MAGNETIC_FIELD
TYPE_CONTROLLER_GYROSCOPE	0x00100004	TYPE_GYROSCOPE
TYPE_CONTROLLER_LIGHT	0x00100005	TYPE_LIGHT
TYPE_CONTROLLER_GRAVITY	0x00100009	TYPE_GRAVITY
TYPE_CONTROLLER_LINEAR_ACCELERATION	0x0010000a	TYPE_LINEAR_ACCELERATION
TYPE_CONTROLLER_ROTATION_VECTOR	0x0010000b	TYPE_ROTATION_VECTOR
TYPE_CONTROLLER_MAGNETIC_FIELD_UNCALIBRATED	0x0010000e	TYPE_MAGNETIC_FIELD_UNCALIBRATED
TYPE_CONTROLLER_GAME_ROTATION_VECTOR	0x0010000f	TYPE_GAME_ROTATION_VECTOR
TYPE_CONTROLLER_GYROSCOPE_UNCALIBRATED	0x00100010	TYPE_GYROSCOPE_UNCALIBRATED
TYPE_CONTROLLER_STATIONARY_DETECT	0x0010001d	TYPE_STATIONARY_DETECT

TYPE_CONTROLLER_MOTION_DETECT	0x0010001e	TYPE_MOTION_DETECT
TYPE_CONTROLLER_ACCELEROMETER_UNCALIBRATED	0x00100023	TYPE_ACCELEROMETER_UNCALIBRATED

Android Sensor API を使用したセンサー制御仕様

センサー優先使用機能に対応しているセンサータイプは、通常センサーとトリガーセンサーに分類されます。トリガーセンサーは以下のセンサータイプが該当し、それ以外のセンサータイプは通常センサーに該当します。

- TYPE_STATIONARY_DETECT
- TYPE_MOTION_DETECT
- TYPE_CONTROLLER_STATIONARY_DETECT
- TYPE_CONTROLLER_MOTION_DETECT

Android Sensor API を利用して、アプリから通常センサーとトリガーセンサーを使用する方法を以下で説明します。

■ 通常センサーの使用

1. モジュールのインポート

通常センサーを使用するために以下のモジュールをインポートします。

```
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
```

2. EventListener の利用

センサーデータを取得するために、Activity や Service に SensorEventListener を implements します。

```
public class SampleActivity extends Activity implements SensorEventListener {
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
    @Override
    public void onSensorChanged(SensorEvent event) {
    }
}
```

3. 使用するセンサーの登録

システムからセンサーサービスを取得し、使用したいセンサーを SensorEventListener に登録します。

なお同時に使用可能なセンサーに特に制限はありませんが、パフォーマンス低下を防ぐために、必要なセンサーのみを登録することを推奨します。

例：TYPE_ACCELEROMETER、TYPE_CONTROLLER_ACCELEROMETER（センサーID：0x00100001）の登録

```
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
```

```
// Register accelerometer
```

```

Sensor acc = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sm.registerListene(this, acc, SensorManager.SENSOR_DELAY_NORMAL);

// Register controller accelerometer
Sensor cacc = sm.getDefaultSensor(0x00100001);
sm.registerListener(this, cacc, SensorManager.SENSOR_DELAY_NORMAL);

```

4. センサーデータの取得

onSensorChanged()でセンサーデータを取得します。

例：TYPE_ACCELEROMETER、TYPE_CONTROLLER_ACCELEROMETER（センサーID：0x00100001）のデータ取得

```

@Override
public void onSensorChanged(SensorEvent event) {
    switch(event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            // Get accelerometer event (Show log)
            Log.d("Sample", "x="+event.values[0]+"y="+event.values[1]+"z="+event.values[2]);
            break;
        case 0x00102001:
            // Get controller accelerometer event (Show log)
            Log.d("Sample", "x="+event.values[0]+"y="+event.values[1]+"z="+event.values[2]);
            break;
    }
}

```

5. センサーの登録解除

アプリでセンサーの使用を止めるときは登録を解除します。

例：TYPE_ACCELEROMETER、TYPE_CONTROLLER_ACCELEROMETERの登録を解除

```

// Unregister accelerometer
sm.unregisterListener(this, acc);

// Unregister controller accelerometer
sm.unregisterListener(this, cacc);

```

■ トリガーセンサーの使用

1. モジュールのインポート

トリガーセンサーを使用するために以下のモジュールをインポートします。

```

import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.hardware.TriggerEvent;
import android.hardware.TriggerEventListener;

```

2. EventListener の利用

センサーを使用する Activity や Service 内に任意のクラスを作成し、TriggerEventListener を extends します。

```
public class SampleActivity extends Activity {  
    public class TriggerListener extends TriggerEventListener {  
        @Override  
        public void onTrigger(TriggerEvent e) {  
        }  
    }  
}
```

3. 使用するセンサーのリクエスト

システムからセンサーサービスを取得し、使用したいセンサーをリクエストします。

例：TYPE_STATIONARY_DETECT のリクエスト

```
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);  
TriggerListener triggerEventListener = new TriggerListener();  
  
// Register stationary detect  
sta = mSensorManager.getDefaultSensor(Sensor.TYPE_STATIONARY_DETECT);  
sm.requestTriggerSensor(triggerEventListener, sta);
```

4. センサーイベント（データ）の取得

onTrigger にセンサーイベントが発行されます。トリガーセンサーはイベントが発行されるとリクエストが解除されるため、再度センサーを使用したい場合はもう一度リクエストしてください。

例) TYPE_STATIONARY_DETECTのデータ取得

```
public class TriggerListener extends TriggerEventListener {  
    @Override  
    public void onTrigger(TriggerEvent e) {  
        switch(e.sensor.getType()) {  
            case Sensor.TYPE_STATIONARY_DETECT:  
                mTextView_staData.setText(String.format("%f", e.values[0]));  
                Log.d("Sample", "Get stationary event : " + e.values[0]);  
                break;  
        }  
    }  
}
```

サンプルコード

アプリが TYPE_ACCELEROMETER、TYPE_CONTROLLER_ACCELEROMETER、TYPE_STATIONARY_DETECT、TYPE_HEADSET_TAP を使用するサンプルコードです。

■ Activity (SampleActivity.java)

```
Package ***;
```



```
import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.view.WindowManager;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.TriggerEvent;
import android.hardware.TriggerEventListener;

public class SampleActivity extends Activity implements SensorEventListener {
    private SensorEventListener mSensorEventListener = null;
    private TriggerEventListener mTriggerEventListener = null;
    private SensorManager mSensorManager = null;

    // Defined custom sensor type
    private static final int TYPE_CONTROLLER_ACCELEROMETER = 0x00100001;
    private static final int TYPE_HEADSET_TAP = 0x00102001;

    private Sensor mAccSensor = null;
    private CheckBox mCheckBox_acc = null;
    private TextView mTextView_accData = null;

    private Sensor mCAccSensor = null;
    private CheckBox mCheckBox_cacc = null;
    private TextView mTextView_caccData = null;

    private Sensor mTapSensor = null;
    private CheckBox mCheckBox_tap = null;
    private TextView mTextView_tapData = null;
    private TextView mTextView_tapEventTime = null;

    private Sensor mStaSensor = null;
    private CheckBox mCheckBox_sta = null;
```

```

private TextView mTextView_staData = null;
private TextView mTextView_staEventTime = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sample);

    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

    mSensorEventListener = this;
    mTriggerEventListener = new TriggerListener();

    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    // TYPE_ACCELEROMETER
    mTextView_accData = (TextView) this.findViewById(R.id.textView_accData);
    mCheckBox_acc = (CheckBox) this.findViewById(R.id.checkBox_acc);
    mCheckBox_acc.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked) {
                mAccSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
                mSensorManager.registerListener(mSensorEventListener, mAccSensor,
SensorManager.SENSOR_DELAY_NORMAL);
            } else {
                mSensorManager.unregisterListener(mSensorEventListener, mAccSensor);
            }
        }
    });

    // TYPE_CONTROLLER_ACCELEROMETER
    mTextView_caccData = (TextView) this.findViewById(R.id.textView_caccData);
    mCheckBox_cacc = (CheckBox) this.findViewById(R.id.checkBox_cacc);
    mCheckBox_cacc.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked) {
                mCAccSensor = mSensorManager.getDefaultSensor(TYPE_CONTROLLER_ACCELEROMETER);
                mSensorManager.registerListener(mSensorEventListener, mCAccSensor,
SensorManager.SENSOR_DELAY_NORMAL);

```

```

        } else {
            mSensorManager.unregisterListener(mSensorEventListener, mCAccSensor);
        }
    }
});

```

```

// TYPE_STATIONARY_DETECT

```

```

mTextView_staData = (TextView) this.findViewById(R.id.textView_staData);
mCheckBox_sta = (CheckBox) this.findViewById(R.id.checkBox_sta);
mTextView_staEventTime = (TextView) this.findViewById(R.id.textView_staTime);
mCheckBox_sta.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            mStaSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_STATIONARY_DETECT);
            mSensorManager.requestTriggerSensor(mTriggerEventListener, mStaSensor);
        }
    }
});

```

```

// TYPE_HEADSET_TAP

```

```

mTextView_tapData = (TextView) this.findViewById(R.id.textView_tapData);
mCheckBox_tap = (CheckBox) this.findViewById(R.id.checkBox_tap);
mTextView_tapEventTime = (TextView) this.findViewById(R.id.textView_tapTime);
mCheckBox_tap.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {

```

```

            mTapSensor = mSensorManager.getDefaultSensor(TYPE_HEADSET_TAP);

```

```

            mSensorManager.registerListener(mSensorEventListener,

```

```

            mTapSensor,

```

```

            SensorManager.SENSOR_DELAY_NORMAL);

```

```

        } else {

```

```

            mSensorManager.unregisterListener(mSensorEventListener, mTapSensor);

```

```

        }

```

```

    }

```

```

});

```

```

}

```

```

@Override

```

```

protected void onPause() {

```

```

    super.onPause();

```

```

    if (mSensorManager != null) {

```

```

        mSensorManager.unregisterListener(this);
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}

@Override
public void onSensorChanged(SensorEvent event) {

    String data = "";
    for(int i = 0; i < event.values.length; i++) {
        data += String.format("%.3f", event.values[i]) + ", ";
    }

    switch(event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            // Show sensor data
            mTextView_accData.setText(data);
            break;
        case TYPE_CONTROLLER_ACCELEROMETER:
            // Show sensor data and
            mTextView_caccData.setText(data);
            break;
        case TYPE_HEADSET_TAP:
            // Show sensor data and time of occurring event
            mTextView_tapData.setText(data);
            mTextView_tapEventTime.setText(getNowTime());
            break;
    }
}

public class TriggerListener extends TriggerEventListener {
    @Override
    public void onTrigger(TriggerEvent e) {
        switch(e.sensor.getType()) {
            case Sensor.TYPE_STATIONARY_DETECT:
                // Show sensor data and time of occurring event
                mTextView_staData.setText(String.format("%.3f", e.values[0]));
                mTextView_staEventTime.setText(getNowTime());
                if(mCheckBox_sta.isChecked()) mCheckBox_sta.setChecked(false);
            }
        }
    }
}

```



```

        android:layout_weight="2"
        android:text="TYPE_ACCELEROMETER" />
    <Space
        android:layout_width="5dip"
        android:layout_height="match_parent" />
    <TextView
        android:id="@+id/textView_accData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="data" />
</LinearLayout>

<!-- TYPE_CONTROLLER_ACCELEROMETER -->
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <CheckBox
        android:id="@+id/checkBox_cacc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="TYPE_CONTROLLER_ACCELEROMETER" />
    <Space
        android:layout_width="5dip"
        android:layout_height="match_parent" />
    <TextView
        android:id="@+id/textView_caccData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="data" />
</LinearLayout>

<Space
    android:layout_width="match_parent"
    android:layout_height="10dip" />

<!-- TYPE_STATIONARY_DETECT -->
<LinearLayout
    android:layout_width="wrap_content"

```

```
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <CheckBox
            android:id="@+id/checkBox_sta"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:text="TYPE_STATIONARY_DETECT" />

        <Space
            android:layout_width="5dip"
            android:layout_height="match_parent" />

        <TextView
            android:id="@+id/textView_staData"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="data" />

        <Space
            android:layout_width="5dip"
            android:layout_height="match_parent" />

        <TextView
            android:id="@+id/textView_staTime"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:text="event time" />
    </LinearLayout>

    <Space
        android:layout_width="match_parent"
        android:layout_height="10dip" />

    <!-- TYPE_HEADSET_TAP -->
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <CheckBox
            android:id="@+id/checkBox_tap"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
        android:layout_weight="2"
        android:text="TYPE_HEADSET_TAP" />
    <Space
        android:layout_width="5dip"
        android:layout_height="match_parent" />
    <TextView
        android:id="@+id/textView_tapData"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="data" />
    <Space
        android:layout_width="5dip"
        android:layout_height="match_parent" />
    <TextView
        android:id="@+id/textView_tapTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="event time" />
</LinearLayout>

<Space
    android:layout_width="match_parent"
    android:layout_height="10dip" />

</LinearLayout>

</ScrollView>

</LinearLayout>
```