

デベロッパーズガイド

Moverio Basic 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.

コンテンツ

Moverio 向けアプリ開発

Moverio 機能の機種別対応表

Android アプリ開発

ディスプレイ制御

センサー制御

カメラ制御

オーディオ制御

デバイス管理

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

Kotlin から Moverio Basic Function SDK を利用する

Android のマルチディスプレイの活用

Windows アプリ開発の概要

ディスプレイ制御(for Windows)

センサー制御(for Windows)

カメラ制御(for Windows)

オーディオ制御(for Windows)

デバイス管理(for Windows)

Moverio 向けアプリ開発の概要

BT-35E などの USB 接続型 Moverio は、Android スマートフォンや Windows パソコンなどと接続し利用することが可能です。

※Android スマートフォンや Windows パソコンの仕様により、Moverio の機能が利用できない場合があります。

※Linux 搭載パソコンでの動作はサポートしていません。

USB 接続型 Moverio 向けアプリ開発は、Android アプリと Windows アプリの開発が可能となります。

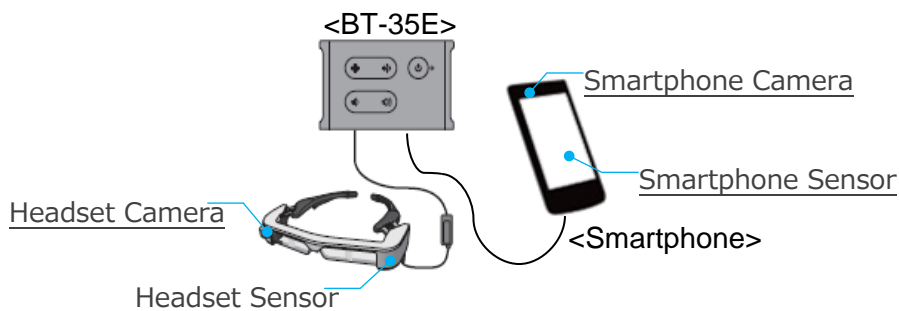


Figure USB 接続型 Moverio の構成

以下のアプリ開発環境をサポートします。

	Android Studio	Unity*1	Visual Studio
Android アプリ	✓	✓	
Windows アプリ			✓

※1 Unity 2018.4.0f1 以降をサポート

Android アプリの開発では、Android 標準のソフトウェア開発キット（Android SDK）と Moverio 専用のソフトウェア開発キット（Moverio Basic Function SDK）を使用することで、Moverio に搭載されているディスプレイ・カメラ・センサー等の制御が可能となります。また、Moverio Basic Function SDK は、BT-35E 以降の後継機種との互換性を維持した API を提供し、Moverio 向けアプリケーションにおける機種間の互換性を担保します。Unity で Android アプリを開発する場合は、Moverio 専用のプラグインを使用することで、Moverio に搭載されているディスプレイ・カメラ・センサー等の制御が可能となります。

Windows アプリの開発では、Windows 標準のソフトウェア開発キットを使用することで、Moverio に搭載されているディスプレイ・カメラ・センサー等の制御が可能となります。

Moverio 向けのアプリ開発の方法は以下を確認してください。

- [Android アプリ開発の概要](#)
- [Unity で Android アプリ開発](#) ※別紙参照
- [Windows アプリ開発の概要](#)
- [Moverio Basic Function SDK の適用範囲](#)

Moverio Basic Function SDK の適用範囲

Moverio Basic Function SDK(旧 Moverio SDK)はバージョンアップにより対応機種を追加します。Moverio Basic Function SDKを使用する際には、必ず最新の Moverio Basic Function SDK を使用してください。

Moverio Basic Function SDK は、BT-350 以前の過去機種との互換性はありません。Moverio Basic Function SDK は Android アプリ開発にのみ適用することができます。また、Android アプリ開発時に、過去機種で提供された SDK と共存することはできますが、BT-350 以前の機種ではそれぞれの機種専用の SDK を使用してください。

	BT-200	BT-2000 BT-2200	BT-300	BT-350	BT-35E	BT-30C	BT-40 BT-40S	BT-45C BT-45CS
Android SDK	✓	✓	✓	✓				
BT-200 SDK	✓							
BT-2000 SDK		✓						
BT-300 SDK			✓	✓				
BT-350 SDK				✓				
Moverio SDK V1.0.0/1.0.1					✓	✓		
MoverioBasic FunctionSDK V1.1.0					✓	✓	✓	
MoverioBasic FunctionSDK V1.2.0					✓	✓	✓*	✓*
MoverioBasic FunctionSDK V1.2.1					✓	✓	✓*	✓*
MoverioBasic FunctionSDK V1.2.2					✓	✓	✓*	✓*

* MoverioBasicFunctionSDK V1.1.0 で提供していた Moverio の専用コントローラー（型番：BO-IC400）の機能は、Moverio Controller Function SDK で提供。

Moverio 機能の機種対応表

Moverio は機種ごとに提供する機能が異なります。

関連する情報を参照してください。

- [ディスプレイ制御機能の機種対応表](#)
- [センサー制御機能の機種対応表](#)
- [カメラ制御機能の機種対応表](#)
- [オーディオ制御機能の機種対応表](#)
- [デバイス管理機能の機種対応表](#)

ディスプレイ制御機能の機種対応表

機種ごとの機能の対応は、下表を参照してください。

機能	BT-35E/30E	BT-30C	BT-40	BT-45C
ディスプレイの明るさの調整（手動）	●	●	●	●
ディスプレイの明るさの調整（自動）	●		●	●
2D/3D 表示モードの切り替え	●	●	●	●
ディスプレイの表示/非表示の設定			●	●
ディスプレイの輻輳の調整			●	●
ディスプレイの自動スリープ設定			●	●
ディスプレイのユーザースリープ設定			●	●

センサー制御機能の機種対応表

機種ごとの機能の対応は、下表を参照してください。

機能	BT-35E/30E	BT-30C	BT-40	BT-45C
加速度センサー制御	●	●	●	●
地磁気センサー制御	●	●	●	●
ジャイロセンサー制御	●	●	●	●
照度センサー制御	●		●	●
重力センサー制御	●	●	●	●
線形加速度センサー制御	●	●	●	●
回転ベクトルセンサー制御	●	●	●	●
地磁気センサー未使用の回転ベクトル			●	●

センサー制御				
未校正の加速度センサー制御			●	●
未校正の地磁気センサー制御			●	●
未校正のジャイロセンサー制御			●	●
ヘッドセット動作検知			●	●
ヘッドセット静止検知			●	●
ヘッドセットタップ検知			●	●

カメラ制御機能の機種対応表

機種ごとの機能の対応は、下表を参照してください。

機能	BT-35E/30E	BT-30C	BT-40	BT-45C
プレビューの表示	●			●
カメラ映像データの取得	●			●
静止画/動画の撮影	●			●
データフォーマット設定	●			●
解像度/フレームレート設定	●			●
露出補正モード設定	●			●
手動露出補正ステップ調整	●			●
オートフォーカスモード設定				●
手動フォーカス調整				●
シャープネス調整	●			
明るさ調整	●			●
ゲイン調整	●			●
ホワイトバランス調整	●			●
電力線周波数設定(50Hz/60Hz)	●			●
電力線周波数設定(Disable)				●
インジケーター制御				●

オーディオ制御機能の機種対応表

機種ごとの機能の対応は、下表を参照してください。

機能	BT-35E/30E	BT-30C	BT-40	BT-45C
イヤホンの音量の調整	●	●		
イヤホンの音量の上限解除設定		●		
オーディオゲイン調整				●
オーディオ入出力デバイスモード制御				●

デバイス管理機能の機種対応表

機種ごとの機能の対応は、下表を参照してください。

機能	BT-35E/30E	BT-30C	BT-40	BT-45C
ヘッドセットシステム状態の通知	●	●	●	●
ヘッドセットシリアル番号の取得			●	●
製品名の取得	●	●	●	●
ヘッドセットシステムバージョンの取得	●	●	●	●
デバイス温度の取得			●	●

Android アプリ開発の概要

Moverio 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以下に記載します。

- [Android SDK の導入](#)
- [Moverio Basic Function SDK の組み込み](#)
- [API リファレンス](#)

Android SDK の導入

以下の項目では 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。（2020 年 9 月時点では、Android Studio 4.0.1）

<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>

注意：ビルドするアプリの targetSdkVersion を 35 とする場合は、Android Studio Koala Feature Drop | 2024.1.2 をお使いください

USB ドライバの設定

アプリケーション開発パソコンと接続するために、ご使用のホスト機器の USB ドライバの設定を行います。USB ドライバの設定は、ご使用のホスト機器の設定手順をご確認ください。

ホスト機器とパソコンとの接続

ADB (Android Debug Bridge)の設定が完了したパソコンとホスト機器の接続について記載します。

ホスト機器の設定

ホスト機器の開発者向けオプションを有効にしてください。次に、USB デバッグを有効にしてください。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/debug/dev-options>

接続確認方法

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 を通しておくとも便利です。

"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 Basic Function SDK の組み込み

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 BasicFunctionSDK_1.2.2.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 24 で定義し、アプリケーションの依存関係に BasicFunctionSDK_1.2.2.aar を追加してください。

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

```
build.gradle(Module: app)
```

```
android {  
    :  
    // API レベル 24 を追加  
    minSdkVersion 24  
    :  
}  
  
dependencies {  
    :  
    // MoverioBasicFunctionSDK を追加  
    implementation files('libs/BasicFunctionSDK_1.2.2.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。

build.gradle の minifyEnabled プロパティを true にした場合の注意事項

Moverio Basic Function SDK を込みこんだアプリケーションで、build.gradle の minifyEnabled プロパティを true にしてビルドする場合、Android Studio のバージョンによっては Moverio Basic Function SDK の機能が正常に動作できない場合があります。minifyEnabled プロパティを true にする場合は、以下のようにアプリケーションの proguard-rules.pro ファイルで、Moverio Basic Function SDK のパッケージに含まれるクラスを難読化・最適化の対象から除外してください。

```
:  
-keep class com.epson.moverio.** {*; }  
:
```

```
proguard-rules.pro(Module: app)
```

ディスプレイ制御の概要

Moverio はシースルー型（透過型）のディスプレイを搭載しています。Moverio Basic Function SDK ではディスプレイの明るさや 2D/3D 表示モードの切り替えなどを制御することができます。ディスプレイの明るさは専用 API からの調整と周辺環境の照度に合わせて自動調整の方法を使用することができます。また、Side-by-Side 方式の 3D コンテンツの表示に対応することが可能です。

関連する情報を参照してください。

- [ディスプレイの明るさの調整](#)
- [2D/3D 表示モードの切り替え](#)
- [ディスプレイの表示/非表示の設定](#)
- [ディスプレイの仮想的な表示距離の調整](#)
- [ディスプレイの自動スリープ設定](#)
- [ディスプレイのユーザースリープ設定](#)
- [ディスプレイ制御機能の機種対応表](#)
- [API リファレンス](#)
- [サンプルコード](#)

ディスプレイの明るさの調整

Moverio はシースルー型のディスプレイを搭載しており、周辺環境の明るさの影響により表示映像の見やすさが変化します。周辺環境が明るい場合はディスプレイの明るさを強くし、周辺環境が暗い場合はディスプレイの明るさを弱くすることで表示映像を見やすくすることができます。

ディスプレイの明るさの調整には、最初に Moverio のディスプレイとの通信を確立します。これを行うには、[DisplayManager](#) クラスのインスタンスを作成して、[open\(\)](#)メソッドを呼び出してください。

※Moverio Basic Function SDK の [DisplayManager](#) クラスの利用の際に、Android 標準の [DisplayManager](#) クラスとの名前衝突を回避するために、`com.epson.moverio.hardware.display.DisplayManager` のインポートをする、もしくは、インスタンス生成時のクラス名を `com.epson.moverio.hardware.display.DisplayManager` としてください。

例) `com.epson.moverio.hardware.display.DisplayManager` のインポート

```
import com.epson.moverio.hardware.display.DisplayManager;
```

例) `com.epson.moverio.hardware.display.DisplayManager` のインスタンス生成

```
public class DisplayActivity extends Activity {  
    private com.epson.moverio.hardware.display.DisplayManager mDisplayManager = null;
```

次に、ディスプレイの明るさ調整モードを手動モード、もしくは、自動モードに設定します。設定には、`setBrightnessMode()`メソッドを使用します。手動モードの場合は `BRIGHTNESS_MODE_MANUAL` 引数を渡して、自動モードの場合は `BRIGHTNESS_MODE_AUTOMATIC` 引数を渡してください。ディスプレイの明るさ調整モードが自動モードの場合は、`setBrightness()`メソッドを使用したディスプレイの明るさの調整はできません。

そして、ディスプレイの明るさを手動で調整するには、`setBrightness()`メソッドを使用してディスプレイの明るさの調整をしてください。調整できるディスプレイの明るさの範囲は、[ディスプレイの明るさの範囲](#)を参照してください。

最後に、ディスプレイの明るさ調整を完了した場合は、必ず Moverio のディスプレイとの通信を解除してください。解除には、`close()`メソッドと `release()`メソッドを使用してください。

```
private DisplayManager mDisplayManager = new DisplayManager(context);
try {
    mDisplayManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mDisplayManager.setBrightnessMode(DisplayManager.BRIGHTNESS_MODE_MANUAL);
mDisplayManager.setBrightness(15);
mDisplayManager.close();
mDisplayManager.release();
```

2D/3D 表示モードの切り替え

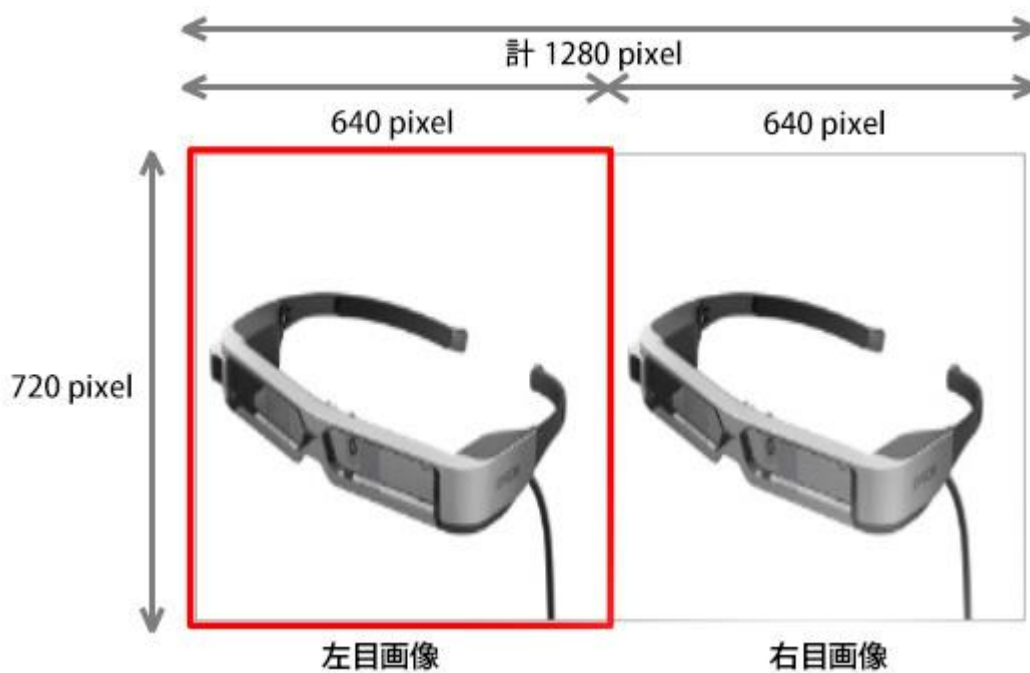
Moverio のディスプレイは、Side-by-Side 方式の 3D コンテンツの表示に対応しています。Moverio で表示する映像が Side-by-Side 方式の 3D コンテンツの場合でアプリから制御する場合は、専用 API を使用してください。

Side-by-Side 方式とは一つの画面に左右の画像を並べて格納する方式です。



Side-by-Side 方式の映像を作成する場合は、映像の左半分と右半分に、それぞれ左目用映像と右目用映像を配置してください。

例えば、1280x720（HD サイズ）の Side-by-Side 方式の映像を作成する場合は、下図のように左目用映像に 640x720 の映像を、右目用映像に 640x720 の映像を配置することで Side-by-Side 方式の映像とすることができます。



ディスプレイの 2D/3D 表示モードの切り替えには、最初に Moverio のディスプレイとの通信を確立します。これを行うには、[DisplayManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出してください。

次に、ディスプレイの 2D/3D 表示モードを 2D 表示モード、もしくは、3D 表示モードに設定します。設定には、[setDisplayMode\(\)](#) メソッドを使用します。2D 表示モードの場合は [DISPLAY_MODE_2D](#) 引数を渡して、3D 表示モードの場合は [DISPLAY_MODE_3D](#) 引数を渡してください。

最後に、2D/3D 表示モードの設定を完了した場合は、必ず Moverio のディスプレイとの通信を解除してください。解除には、[close\(\)](#) メソッドと [release\(\)](#) メソッドを使用してください。

```
private DisplayManager mDisplayManager = new DisplayManager(context);
try {
    mDisplayManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mDisplayManager.setDisplayMode(DisplayManager.DISPLAY_MODE_3D);
mDisplayManager.close();
mDisplayManager.release();
```

ディスプレイの表示/非表示の設定

Moverio に表示されている映像を一時的に非表示にしたい場合は、専用 API を使用してください。専用 API の対応機種は[ディスプレイ制御機能の機種対応表](#)を参照してください。



ディスプレイの表示/非表示の設定には、最初に Moverio のディスプレイとの通信を確立します。これを行うには、`DisplayManager` クラスのインスタンスを作成して、`open()` メソッドを呼び出してください。

次に、ディスプレイを表示、もしくは、非表示に設定します。設定には、`setDisplayState()` メソッドを使用します。ディスプレイを表示する場合は `DISPLAY_STATE_ON` 引数を渡して、ディスプレイを非表示にする場合は `DISPLAY_STATE_OFF` 引数を渡してください。

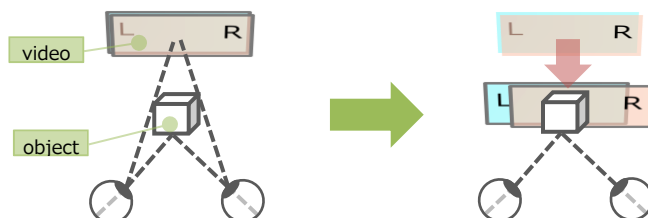
最後に、ディスプレイの表示/非表示の設定を完了した場合は、必ず Moverio のディスプレイとの通信を解除してください。解除には、`close()` メソッドと `release()` メソッドを使用してください。

```
private DisplayManager mDisplayManager = new DisplayManager(context);
try {
    mDisplayManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mDisplayManager.setDisplayState (DisplayManager.DISPLAY_STATE_OFF);
mDisplayManager.close();
mDisplayManager.release();
```

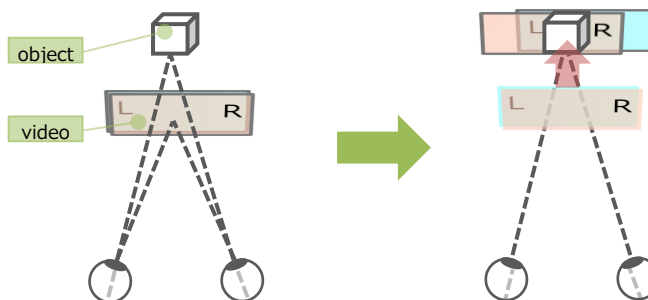
ディスプレイの仮想的な表示距離の調整

Moverio のディスプレイの表示距離を専用 API で調整することができます。専用 API は左右の表示映像を水平方向にシフトする機能があります。水平方向のシフト量に応じてディスプレイの輻輳を調整できます。

例えば、実際のオブジェクトがディスプレイの表示距離よりも近い位置にある場合は、下図のように左右の映像を内側にシフトします。そうすることで、ディスプレイの仮想的な表示距離を近くにすることができます。



その他に、実際のオブジェクトがディスプレイの表示距離よりも遠い位置にある場合は、下図のように左右の映像を外側にシフトします。そうすることで、ディスプレイの仮想的な表示距離を遠くにすることができます。



ただし、この機能を利用することで左右の映像をシフトさせるため、左右の映像の端が途切れてしまいます。そのため、アプリケーションで映像のシフト量に応じた画面構成の配慮が必要になります。ディスプレイの仮想的な表示距離と映像のシフト量の関係性は、[ディスプレイの仮想的な表示距離の調整の範囲と水平方向のシフト量](#)を参照してください。

ディスプレイの仮想的な表示距離の調整には、最初に Moverio のディスプレイとの通信を確立します。これを行うには、`DisplayManager` クラスのインスタンスを作成して、`open()`メソッドを呼び出してください。

次に調整量のステップを設定します。設定には、`setScreenHorizontalShiftStep()`メソッドを使用します。調整量のステップは、[ディスプレイの仮想的な表示距離の調整の範囲と水平方向のシフト量](#)を参照してください。

最後に、調整を完了した場合は、必ず Moverio のディスプレイとの通信を解除してください。解除には、`close()`メソッドと `release()`メソッドを使用してください。

```
private DisplayManager mDisplayManager = new DisplayManager(context);
try {
    mDisplayManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mDisplayManager.setScreenHorizontalShiftStep(39);
mDisplayManager.close();
```



```
mDisplayManager.release();
```

ディスプレイの自動スリープ設定

Moverio のディスプレイの自動スリープ設定を専用 API で設定することができます。専用 API の対応機種は[ディスプレイ制御機能の機種対応表](#)を参照してください。

ディスプレイの自動スリープ設定を有効化した場合は、Moverio を脱着して机上などに置くと、自動的にディスプレイを消灯します。また、Moverio を装着すると自動的にディスプレイを表示します。この機能を有効にすることで、Moverio を装着していない場合の消費電力を低減させることができます。

ディスプレイの自動スリープ設定には、最初に Moverio のディスプレイとの通信を確立します。これを行うには、`DisplayManager` クラスのインスタンスを作成して、`open()` メソッドを呼び出してください。

次に、ディスプレイの自動スリープ設定を有効化、もしくは、無効化に設定します。設定には、`setDisplayAutoSleepEnabled()` メソッドを使用します。設定を有効化する場合は `DISPLAY_AUTO_SLEEP_ENABLE` 引数を渡して、設定を無効化する場合は `DISPLAY_AUTO_SLEEP_DISABLE` 引数を渡してください。

最後に、設定を完了した場合は、必ず Moverio のディスプレイとの通信を解除してください。解除には、`close()` メソッドと `release()` メソッドを使用してください。

```
private DisplayManager mDisplayManager = new DisplayManager(context);
try {
    mDisplayManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mDisplayManager.setDisplayAutoSleepEnabled(DisplayManager.DISPLAY_AUTO_SLEEP_ENABLE);
mDisplayManager.close();
mDisplayManager.release();
```

ディスプレイのユーザースリープ設定

Moverio のディスプレイのユーザースリープ設定を専用 API で設定することができます。専用 API の対応機種は[ディスプレイ制御機能の機種対応表](#)を参照してください。

ディスプレイのユーザースリープ設定を有効化した場合は、Moverio の側面を軽くタップするとディスプレイを消灯します。また、再度 Moverio の側面を軽くタップするとディスプレイを表示します。この機能は、Moverio の装着者が前面の映像を即座に消したい時に利用できます。

ディスプレイのユーザースリープ設定には、最初に Moverio のディスプレイとの通信を確立します。これを行うには、[DisplayManager](#) クラスのインスタンスを作成して、[open\(\)](#)メソッドを呼び出してください。

次に、ディスプレイのユーザースリープ設定を有効化、もしくは、無効化に設定します。設定には、[setDisplayUserSleepEnabled\(\)](#)メソッドを使用します。設定を有効化する場合は [DISPLAY_USER_SLEEP_ENABLE](#) 引数を渡して、設定を無効化する場合は [DISPLAY_USER_SLEEP_DISABLE](#) 引数を渡してください。

最後に、設定を完了した場合は、必ず Moverio のディスプレイとの通信を解除してください。解除には、[close\(\)](#)メソッドと [release\(\)](#)メソッドを使用してください。

```
private DisplayManager mDisplayManager = new DisplayManager(context);
try {
    mDisplayManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mDisplayManager.setDisplayUserSleepEnabled(DisplayManager.DISPLAY_USER_SLEEP_ENABLE);
mDisplayManager.close();
mDisplayManager.release();
```

シースルーを活かした映像の作り方

Moverio はシースルー型のディスプレイを搭載しており、Moverio の装着者は実空間で見えている景色を見ながら表示される映像を重ね合わせてみることができます。実空間で見えている景色に Moverio で表示したいオブジェクトの映像を重ね合わせて見る場合は、表示したいオブジェクトの背景を黒にすることで、シースルーを活かした映像を作ることができます。



センサー制御の概要

Moverio は動き、向き、環境を検知する様々なセンサーをヘッドセットに搭載しています。Moverio Basic Function SDK では、これらのセンサーの生のデータを取得することができます。センサーデータを使用することで、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。

関連する情報を参照してください。

- [センサーの種類](#)
- [センサーの軸](#)
- [センサーデータの取得](#)
- [センサー制御機能の機種対応表](#)
- [API リファレンス](#)
- [サンプルコード](#)

センサーの種類

Moverio は様々な種類のセンサーを搭載しています。センサーの中には、ハードウェアベースのセンサーとソフトウェアベースのセンサーがあります。ハードウェアベースのセンサーには、加速度、地磁気、角速度、環境照度などがあります。ソフトウェアベースのセンサーは、単体もしくは複数のハードウェアベースのセンサーの出力値を使用した仮想のセンサーです。ソフトウェアベースのセンサーには、回転ベクトルや重力などの推測が可能なものがあります。

センサー	種類	説明	使用例
TYPE_ACCELEROMETER	ハードウェア	重力を含む Moverio の加速度を 3 軸 (x、y、z) の加速度[m/s ²]で測定します。	モーション検知 (傾きなど)
TYPE_MAGNETIC_FIELD	ハードウェア	周囲の地磁気を 3 軸 (x、y、z) の地磁気 [μT]で測定します。	方位検知
TYPE_GYROSCOPE	ハードウェア	Moverio の角速度を 3 軸 (x、y、z) の角速度[rad/s]で測定します。	モーション検知 (回転など)
TYPE_LIGHT	ハードウェア	周囲の照度[lx]を測定します。	周囲の照度に合わせたディスプレイの明るさ調整
TYPE_GRAVITY	ソフトウェア	重力を 3 軸 (x、y、z) の加速度[m/s ²]で測定します。	モーション検知 (傾きなど)
TYPE_LINEAR_ACCELERATION	ソフトウェア	重力を除いた 3 軸 (x、y、z) の加速度 [m/s ²]で測定します。	タップ、歩行の検出などへの応用
TYPE_ROTATION_VECTOR	ソフトウェア	Moverio の向きを回転ベクトルで測定します。	ヘッドトラッキングなど
TYPE_MAGNETIC_FIELD_UNC	ソフトウェア	周囲の地磁気を未校正の 3 軸 (x、y、z)	方位検知、ヘッドトラッ

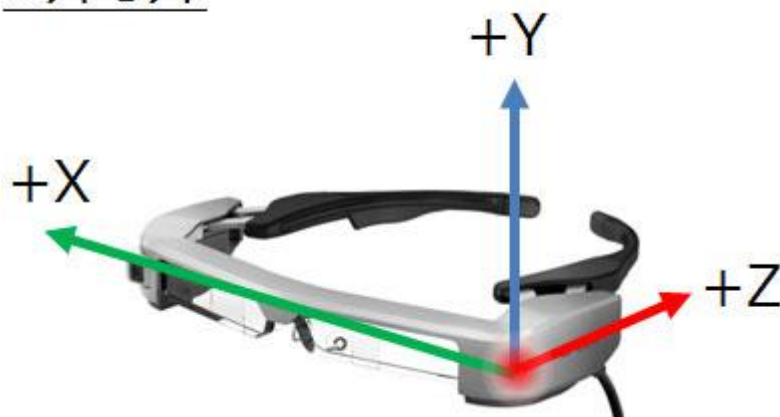
ALIBRATED		の地磁気[μT]と校正情報の 3 軸 (x_bias , y_bias , z_bias) で測定します。	キングへの応用
TYPE_GAME_ROTATION_VECTOR	ソフトウェア	Moverio の向きを、地磁気を使用しない回転ベクトルで測定します。	ヘッドトラッキングなど
TYPE_GYROSCOPE_UNCALIBRATED	ソフトウェア	Moverio の角速度を未校正の 3 軸 (x , y , z) の角速度[rad/s]と推定ドリフトの 3 軸 (x_bias , y_bias , z_bias) で測定します。	モーション検知 (回転など)、ヘッドトラッキングへの応用
TYPE_STATIONARY_DETECT	ソフトウェア	Moverio の静止を検知します。	ヘッドセット着脱検知など
TYPE_MOTION_DETECT	ソフトウェア	Moverio の動作を検知します。	ヘッドセット着脱検知など
TYPE_ACCELEROMETER_UNCALIBRATED	ソフトウェア	重力を含む Moverio の加速度を未校正の 3 軸 (x , y , z) の加速度[m/s^2]と推定バイアス補償の 3 軸 (x_bias , y_bias , z_bias) で測定します。	モーション検知 (傾きなど)、ヘッドトラッキングへの応用
TYPE_HEADSET_TAP_DETECT	ソフトウェア	ヘッドセットのタップを検知します。	ヘッドセットタップ検知

詳細は [API リファレンス](#) を参照してください。

センサーの軸

Moverio のセンサーは、Android 標準のセンサーの座標系と同じです。Moverio 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。

ヘッドセット



センサーデータの取得

Moverio Basic Function SDK を使用すると、Moverio に搭載されている各種センサーを使用することができます。

センサーデータの取得には、最初に Moverio のセンサーとの通信を確立します。これを行うには、[SensorManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出してください。その際に、センサーの種類とセンサーデータのリスナーのインスタンスを引数で渡してください。

※[SensorManager#open](#) は、Moverio とスマートフォンを USB 接続してから 10 秒以上経過後に、Moverio に映像が表示されていることを確認の上、実行してください。10 秒以内に [SensorManager#open](#) を実行した場合は、正常に通信を確立できない場合があります。

※Moverio Basic Function SDK の [SensorManager](#) クラスの利用の際に、Android 標準の [SensorManager](#) クラスとの名前衝突を回避するために、`com.epson.moverio.hardware.sensor.SensorManager` のインポートをする、もしくは、インスタンス生成時のクラス名を `com.epson.moverio.hardware.sensor.SensorManager` としてください。

例) `com.epson.moverio.hardware.sensor.SensorManager` のインポート

```
import com.epson.moverio.hardware.sensor.SensorManager;
```

例) `com.epson.moverio.hardware.sensor.SensorManager` のインスタンス生成

```
public class SensorActivity extends Activity implements SensorDataListener {  
    private com.epson.moverio.hardware.sensor.SensorManager mSensorManager = null;
```

対応するセンサーの種類はモデルによって異なります。対応するセンサーの種類は、[getSupportedSensorList\(\)](#) メソッドで取得し、確認してください。

```
@Override  
public final void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    mSensorManager = new SensorManager(this);  
    // Get supported sensor list.  
    List<Integer> sensorList = mSensorManager.getSupportedSensorList();  
    Log.v("Sample", sensorList.toString());  
}
```

センサーデータは、登録されたリスナーの [onSensorDataChanged\(\)](#) メソッドで随時取得できます。[onSensorDataChanged\(\)](#) メソッドは、非常に高速に呼び出されます。アプリケーションで効率的なセンサー利用を実現するために、[onSensorDataChanged\(\)](#) メソッド内では、可能な限り時間のかかる処理などを実行しないようにしてください。最後に、センサーデータの取得を終了させる場合は、必ず Moverio のセンサーとの通信を解除してください。解除には、[close\(\)](#) メソッドと [release\(\)](#) メソッドを使用してください。

センサーデータは、[SensorData](#) クラスのインスタンスとして取得できます。[SensorData](#) クラスの詳細は、API リファレンスを参照してください。

```
public class SensorActivity extends Activity implements SensorDataListener {  
    private SensorManager mSensorManager = null;
```

```

@Override
public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mSensorManager = new SensorManager(this);
}

@Override
protected void onResume() {
    super.onResume();
    try {
        mSensorManager.open(SensorManager.TYPE_ACCELEROMETER, this);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
protected void onPause() {
    super.onPause();
    mSensorManager.close(this);
    mSensorManager.release();
}

@Override
public void onSensorDataChanged(SensorData data) {
    // Do something with this sensor value.
}
}

```

センサー制御の注意事項

Moverio Basic Function SDK を使用したセンサー制御は、単一のアプリでのみ利用することができます。別のアプリでセンサー制御を利用する場合は、必ずセンサー制御を利用しているアプリでセンサー制御の利用を停止してから、利用してください。Android 端末で同時に複数のセンサーを利用するとセンサーデータの取得頻度が低下する場合があります。

カメラ制御の概要

注意：Moverio Basic Function SDK のカメラ機能は、Android 10 のスマートフォンでは正常に動作しない場合があります。カメラ機能を利用する場合は、以下の動作条件を必ず確認してください。

	targetSdkVersion 28 以上	targetSdkVersion 27 以下
Android 11	動作可能	動作可能
Android 10	動作不可	動作可能
Android 9 以下	動作可能	動作可能

また、2021年11月以降に Google Play に登録するアプリケーションは、targetSdkVersion を 30 以上にする必要があります。

参考：<https://developer.android.com/distribute/best-practices/develop/target-sdk>

Moverio Basic Function SDK のカメラ機能を組み込んだアプリケーションを Google Play でリリースするためには、アプリケーションの対象とする Android のバージョンを 9 以下とする、もしくは、対象とする Android のバージョンを 11 とする必要があります。

注意：CaptureStateCallback クラスは、廃止を予定しています。Version 1.1.0 以降は、必ず CaptureStateCallback2 クラスを利用してください。

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います。Moverio Basic Function SDK では、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。

関連する情報を参照してください。

- [カメラスペック](#)
- [プレビューの表示](#)
- [カメラ映像データの取得](#)
- [静止画/動画の撮影](#)
- [カメラプロパティの変更](#)
- [データフォーマット](#)
- [パーミッション要求](#)
- [SD カードに静止画/動画を保存する方法](#)
- [カメラ制御の注意事項](#)
- [カメラ制御機能の機種対応表](#)
- [API リファレンス](#)
- [サンプルコード](#)

カメラスペック

項目	BT-35E	BT-45C
有効画素数	500 万画素	800 万画素
データフォーマット	RGB565/ARGB8888	RGB565/ARGB8888/YUY2/H.264
[データフォーマット] 解像度/フレームレート	[RGB565/ARGB8888] 640x480, 60fps/30fps/15fps 1280x720, 60fps/30fps/15fps 1920x1080, 30fps/15fps 2592x1944, 15fps	[RGB565/ARGB8888] 640x480, 60fps/30fps 1280x720, 60fps/30fps 1920x1080, 30fps 1600x1200, 30fps 2560x1440, 20fps 3264x2448, 10fps
	-	[YUY2] 640x480, 30fps 1280x720, 10fps 1920x1080, 5fps 1600x1200, 5fps 2560x1440, 2fps 3264x2448, 1fps
	-	[H.264] 640x480, 60fps/30fps 1280x720, 60fps/30fps 1920x1080, 30fps 1600x1200, 30fps 2560x1440, 30fps 3264x2448, 30fps
露出補正モード	auto/manual	auto/manual
手動露出補正ステップ	-5 ~ +5	-7 ~ +5
オートフォーカスモード	off	auto/manual
手動フォーカス	-	+30 ~ +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(6000K)/daylight(5500K) /fluorescent(4200K)/incandescent(3200K) /twilight(3500K)	auto /cloudy_daylight(6000K)/daylight(5500K) /fluorescent(4200K)/incandescent(3200K) /twilight(3500K)
電力線周波数	50Hz/60Hz	50Hz/60Hz/Disable
インジケーターモード	auto	auto/on/off

プレビューの表示

注意：データフォーマットが H.264 の場合は、プレビューの表示はできません。

注意：Surface クラスのインスタンスを用いてプレビュー表示する場合は、[Surface クラスのインスタンスを引数としたプレビューの表示を参照してください。](#)

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラのプレビューを表示することができます。

プレビューの表示には、最初に Moverio のカメラとの通信を確立します。これを行うには、`CameraManager` クラスのインスタンスを作成して、`open()` メソッドを呼び出します。その際に、プレビュー表示に用いる `SurfaceView` の `SurfaceHolder` を引数で渡します。次に、`CameraDevice` クラスの `startCapture()` メソッドを呼び出し、カメラ映像データの取得を開始します。そして、`CameraDevice` クラスの `startPreview()` メソッドを呼び出すことで、`open()` メソッドの引数で渡した `SurfaceHolder` の `SurfaceView` にプレビューの映像が表示されます。

※Moverio Basic Function SDK の `CameraManager` クラスの利用の際に、Android 標準の `CameraManager` クラスとの名前衝突を回避するために、`com.epson.moverio.hardware.camera.CameraManager` のインポートをする、もしくは、インスタンス生成時のクラス名を `com.epson.moverio.hardware.camera.CameraManager` としてください。

例) `com.epson.moverio.hardware.camera.CameraManager` のインポート

```
import com.epson.moverio.hardware.camera.CameraManager;
```

例) `com.epson.moverio.hardware.camera.CameraManager` のインスタンス生成

```
public class CameraActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {
    private com.epson.moverio.hardware.camera.CameraManager mCameraManager = null;
```

プレビューの表示を停止させたい場合は、`stopPreview()` メソッドを呼び出します。カメラの利用を終了する場合は、`stopCapture()` メソッドを呼び出してカメラ映像データの取得を停止させ、`close()` メソッドと `release()` メソッドを呼び出してカメラとの通信を切断します。

```
public class CameraActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {
    private CameraManager mCameraManager = null;
    private CameraDevice mCameraDevice = null;
    private SurfaceView mSurfaceView = null;
    private PermissionHelper mPermissionHelper = null;

    private CaptureStateCallback2 mCaptureStateCallback2 = new CaptureStateCallback2() {

        @Override
        public void onCameraOpened() {
            mCameraDevice.startCapture();
        }
    }
```

```

@Override
public void onCaptureStarted() {
    mCameraDevice.startPreview();
}

@Override
public void onPreviewStopped() {
    mCameraDevice.stopCapture();
}

@Override
public void onCaptureStopped() {
    mCameraManager.close(mCameraDevice);
}

    :
    省略
    :
};

@Override
public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mPermissionHelper = new PermissionHelper(this);
    mSurfaceView = (SurfaceView) findViewById(R.id.surfaceView);
    mCameraManager = new CameraManager(this);
    try {
        mCameraDevice = mCameraManager.open(mCaptureStateCallback2, null,
                                                mSurfaceView.getHolder());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
protected void onPause() {
    super.onPause();
    mCameraDevice.stopPreview();
}

@Override
protected void onDestroy() {

```

```
        super.onDestroy();
        mCameraManager.release();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

カメラ映像データの取得

注意：ByteBuffer 形式でカメラ映像データを取得する場合は、[ByteBuffer 形式でのカメラデータの取得](#)を参照してください。

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラの映像データを取得することができます。

カメラ映像データの取得は、最初に Moverio のカメラとの通信を確立します。これを行うには、[CameraManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出します。その際に、データを受信する [CaptureDataCallback](#) のインスタンスを引数で渡します。そして、[CameraDevice](#) クラスの [startCapture\(\)](#) メソッドを呼び出すことで、[open\(\)](#) メソッドの引数で渡した [CaptureDataCallback](#) のインスタンスでデータを受信することができます。

カメラ映像データの取得を停止させたい場合は、[stopCapture\(\)](#) メソッドを呼び出します。カメラの利用を終了する場合は、[close\(\)](#) メソッドと [release\(\)](#) メソッドを呼び出してカメラとの通信を切断します。

カメラ映像データを取得している状態でアプリをバックグラウンドに移行した場合は、カメラ映像データの取得が停止します。カメラ映像データの取得を再開させるためには、再度 [startCapture\(\)](#) メソッドを呼び出します。

```
public class CameraActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {

    private CameraManager mCameraManager = null;
    private CameraDevice mCameraDevice = null;
    private SurfaceView mSurfaceView = null;
    private PermissionHelper mPermissionHelper = null;

    private CaptureDataCallback mCaptureDataCallback = new CaptureDataCallback() {
        @Override
        public void onCaptureData(long timestamp, byte[] data) {
            // Do something with this camera data.
        }
    };

    private CaptureStateCallback2 mCaptureStateCallback2 = new CaptureStateCallback2() {

        @Override
        public void onCameraOpened() {
            mCameraDevice.startCapture();
        }

        @Override
        public void onCaptureStopped() {
            mCameraManager.close(mCameraDevice);
        }

        :
        省略
    };
}
```

```

        :

};

@Override
public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mPermissionHelper = new PermissionHelper(this);
    mCameraManager = new CameraManager(this);
    try {
        mCameraDevice = mCameraManager.open(mCaptureStateCallback2, mCaptureDataCallback,
                                                mSurfaceView.getHolder());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
protected void onPause() {
    super.onPause();
    mCameraDevice.stopCapture();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    mCameraManager.release();
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
}

```

静止画/動画の撮影

注意：データフォーマットが H.264 の場合は、静止画/動画の撮影はできません。

注意：静止画/動画の撮影を実行する場合は、必ず以下を参照してください。

<https://developer.android.com/training/data-storage/use-cases#opt-out-scoped-storage>

<https://developer.android.com/about/versions/11/privacy/storage>

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラの映像を静止画や動画として撮影することができます。

静止画/動画の撮影は、最初に Moverio のカメラとの通信を確立します。これを行うには、`CameraManager` クラスのインスタンスを作成して、`open()` メソッドを呼び出します。その際に、静止画撮影の完了や動画撮影の開始/終了のステータス通知を受信する `CaptureStateCallback2` のインスタンスを引数で渡します。次に、`CameraDevice` クラスの `startCapture()` メソッドを呼び出します。

静止画撮影する場合は、`CameraDevice` クラスの `takePicture()` メソッドを呼び出します。静止画撮影が完了した際に、`open()` メソッドの引数で渡した `CaptureStateCallback2` のインスタンスの `onPictureCompleted()` メソッドで静止画撮影完了の通知を受信することができます。

動画撮影を開始する場合は `CameraDevice` クラスの `startRecord()` メソッドを呼び出し、動画撮影を終了する場合は `CameraDevice` クラスの `stopRecord()` メソッドを呼び出します。動画撮影の開始/終了した際に、`open()` メソッドの引数で渡した `CaptureStateCallback2` のインスタンスの `onRecordStarted()` メソッドや `onRecordStopped()` メソッドで動画撮影の開始/終了の通知を受信することができます。動画撮影は最大 2 時間実行できます。動画撮影の時間が 2 時間を超えた場合、強制的に動画撮影を終了します。また、動画撮影中にストレージ容量が 10% を下回った場合も強制的に動画撮影を終了します。`stopRecord()` メソッドは Activity ライフサイクルの `onStop/onDestroy` などのメソッド内部での実行は、エラーとなります。

カメラの利用を終了する場合は、`stopCapture()` メソッドを呼び出してカメラ映像データの取得を停止し、`close()` メソッドと `release()` メソッドを呼び出してカメラとの通信を切断してください。

カメラプロパティの変更

注意：カメラの露出調整機能は、自動露出（AE）を利用してください。

自動露出（AE）でカメラ映像が期待通りに取得できない場合のみ、手動露出補正モードを利用してください。

注意：解像度を 640x480 に設定し、環境照度が明るい環境で露出設定を明るく設定すると、カメラ映像データが取得できなくなる場合があります。

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラの各種プロパティを変更することができます。

カメラプロパティの変更は、最初に Moverio のカメラとの通信を確立します。これを行うには、[CameraManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出します。その際に、プレビュー表示に用いる [SurfaceView](#) の [SurfaceHolder](#) からデータを受信する [CaptureDataCallback](#) のインスタンスを引数で渡します。次に、[CameraDevice](#) クラスの [getProperty\(\)](#) メソッドを呼び出し、現状のカメラプロパティを [CameraProperty](#) クラスのインスタンスとして取得します。各種カメラプロパティの変更は、[CameraProperty](#) クラスのインスタンスの [setXXX\(\)](#) メソッドを呼び出します。

例えば、カメラの露出ステップを変更する場合は、[CameraProperty](#) クラスの [setExposureStep\(\)](#) メソッドを呼び出します。その際に、変更後のカメラの露出ステップを引数で渡します。

そして、[CameraDevice](#) クラスの [setProperty\(\)](#) メソッドを呼び出し、変更後のカメラプロパティをカメラ映像に反映します。その際に、変更後のカメラプロパティである [CameraProperty](#) クラスのインスタンスを引数で渡します。

カメラプロパティは一部を除き [startCapture\(\)](#) メソッドの呼び出しによるカメラ映像のキャプチャ後でも変更可能です。

項目	キャプチャ前	キャプチャ後
データフォーマット	変更可能	変更不可
解像度/フレームレート	変更可能	変更不可
露出補正モード	変更可能	変更可能
手動露出補正ステップ	変更可能	変更可能
オートフォーカスモード	変更可能	変更可能
手動フォーカス	変更可能	変更可能
シャープネス	変更可能	変更可能
明るさ	変更可能	変更可能
ゲイン	変更可能	変更可能
ホワイトバランス	変更可能	変更可能
電力線周波数	変更可能	変更可能
インジケータモード	変更可能	変更可能

手動露出補正ステップは、利用環境の環境照度[lx]に応じて変更することができます。ただし、高い環境照度に対応した露出補正ステップは、高い環境照度に対応するためにカメラの露光時間を長くしているため、カメラ映像のフレームレートにも影響を与える場合があります。詳細は下表に記載します。

手動露出補正ステップ	BT-35E		BT-45C	
	環境照度[lx]	最大フレームレート[fps]	環境照度[lx]	最大フレームレート[fps]
+5	50	5.0	50	9.9
+4	100	10.0	100	20.0
+3	150	22.6	150	30.0
+2	200	30.1	200	40.0
+1	500	45.2	500	60.0 *1
0 (=default)	750	60.0 *1	750	60.0 *1
-1	1000	60.0 *1	1000	60.0 *1
-2	1500	60.0 *1	1500	60.0 *1
-3	2000	60.0 *1	2000	60.0 *1
-4	4500	60.0 *1	4500	60.0 *1
-5	9000	60.0 *1	9000	60.0 *1
-6	Not support	Not support	25000	60.0 *1
-7	Not support	Not support	50000	60.0 *1

*1 : Moverio のカメラ映像のフレームレートは、最大 60[fps]となります。また、フレームレートはホスト機器の処理性能などにより低下する場合があります。

データフォーマット

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラの出力するデータフォーマットを変更することができます。データフォーマットの変更は、[カメラプロパティの変更](#)を参照してください。

RGB565/ARGB8888/YUY2 は、Android の ImageFormat の仕様に準拠する。

<https://developer.android.com/reference/android/graphics/ImageFormat>

H.264 は、ITU-T の仕様に準拠する。

<https://www.itu.int/rec/T-REC-H.264>

Surface クラスのインスタンスを引数としたプレビューの表示

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラのプレビューを表示することができます。

プレビューの表示には、最初に Moverio のカメラとの通信を確立します。これを行うには、`CameraManager` クラスのインスタンスを作成して、`open()` メソッドを呼び出します。その際に、プレビュー表示に用いる `SurfaceView` の `SurfaceHolder` を null で渡します。次に、`CameraDevice` クラスの `setPreviewSurface()` メソッドの引数に `Surface` のインスタンスを指定して実行します。その後、`CameraDevice` クラスの `startCapture()` メソッドを呼び出し、カメラ映像データの取得を開始します。そして、`CameraDevice` クラスの `startPreview()` メソッドを呼び出すことで、`setPreviewSurface()` メソッドの引数で渡した `SurfaceHolder` の `SurfaceView` にプレビューの映像が表示されます。

プレビューの表示を停止させたい場合は、`stopPreview()` メソッドを呼び出します。カメラの利用を終了する場合は、`stopCapture()` メソッドを呼び出してカメラ映像データの取得を停止させ、`close()` メソッドと `release()` メソッドを呼び出してカメラとの通信を切断します。

```
public class CameraActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {

    private CameraManager mCameraManager = null;
    private CameraDevice mCameraDevice = null;
    private SurfaceView mSurfaceView = null;
    private PermissionHelper mPermissionHelper = null;

    private CaptureStateCallback2 mCaptureStateCallback2 = new CaptureStateCallback2() {
        @Override
        public void onCameraOpened() {
            // set Surface before CameraDevice#startCapture.
            mCameraDevice.setPreviewSurface(mSurfaceView.getHolder().getSurface());

            mCameraDevice.startCapture();
        }
        :
        省略
        :
    };

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mPermissionHelper = new PermissionHelper(this);
        mSurfaceView = (SurfaceView) findViewById(R.id.surfaceView);
        mCameraManager = new CameraManager(this);
        try {
            // Set the third argument to null.
            mCameraDevice = mCameraManager.open(mCaptureStateCallback2, null, null);
```

```

    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
}

```

ByteBuffer 形式でのカメラデータの取得

Moverio Basic Function SDK を使用すると、Moverio に搭載されているカメラの映像データを取得することができます。

カメラ映像データの取得は、最初に Moverio のカメラとの通信を確立します。これを行うには、[CameraManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出します。その際に、データを受信する [CaptureDataCallback](#) を null で渡します。次に、[CameraDevice](#) クラスの [setCaptureDataCallback\(\)](#) メソッドの引数に [CaptureDataCallback2](#) のインスタンスを指定して実行します。そして、[CameraDevice](#) クラスの [startCapture\(\)](#) メソッドを呼び出すことで、[setCaptureDataCallback\(\)](#) メソッドの引数で渡した [CaptureDataCallback2](#) のインスタンスでデータを受信することができます。

カメラ映像データの取得を停止させたい場合は、[stopCapture\(\)](#) メソッドを呼び出します。カメラの利用を終了する場合は、[close\(\)](#) メソッドと [release\(\)](#) メソッドを呼び出してカメラとの通信を切断します。

カメラ映像データを取得している状態でアプリをバックグラウンドに移行した場合は、カメラ映像データの取得が停止します。カメラ映像データの取得を再開させるためには、再度 [startCapture\(\)](#) メソッドを呼び出します。

```

public class CameraActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {

    private CameraManager mCameraManager = null;
    private CameraDevice mCameraDevice = null;
    private PermissionHelper mPermissionHelper = null;

    private CaptureStateCallback2 mCaptureStateCallback2 = new CaptureStateCallback2() {
        @Override
        public void onCameraOpened() {
            // set instance of CaptureDataCallback2 before CameraDevice#startCapture.
            mCameraDevice.setCaptureDataCallback(mCaptureDataCallback2);
        }
    }
}

```

```

        mCameraDevice.startCapture();
    }

    :
    省略
    :

};

private CaptureDataCallback2 mCaptureDataCallback2 =new CaptureDataCallback2() {
    @Override
    public void onCaptureData(long timestamp, ByteBuffer data) {
        // Do something with this camera data.
    }
};

@Override
public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mPermissionHelper = new PermissionHelper(this);
    mCameraManager = new CameraManager(this);
    try {
        // Set the second argument to null.
        mCameraDevice = mCameraManager.open(mCaptureStateCallback2, null, null);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
}

```

カメラのパーミッション要求

Moverio Basic Function SDK を用いて開発したアプリケーションを Android 端末で初めて動作させる際に、Android 端末の利用者に対しパーミッション要求を行います。要求するパーミッションは以下の通りです。

項目	要求タイミング
カメラ (Android9 以降)	CameraManager#open() メソッドの呼び出し時
USB 通信	CameraManager#open() メソッドの呼び出し時
静止画/動画撮影時のストレージへのファイル書き込み (Android10 以前)	CameraDevice#startCapture() メソッドの呼び出し時
録画時の録音許可	CameraDevice#startCapture() メソッドの呼び出し時

アプリケーションでは、ActivityCompat.OnRequestPermissionsResultCallback インターフェースを実装し、利用者がパーミッションに許可したことを受信します。その際に、PermissionHelper クラスの onRequestPermissionsResult() メソッドを呼び出し、Moverio Basic Function SDK へ利用者の許可を通知します。Moverio Basic Function SDK へ利用者の許可が通知されるとカメラ機能の利用が可能となります。

```
public class CameraActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {
    private PermissionHelper mPermissionHelper = null;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPermissionHelper = new PermissionHelper(this);
        :
        :
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

SD カードに静止画/動画を保存する方法

静止画/動画の撮影結果を SD カードに保存するためには、SD カード内部のアプリケーション用データ保存領域へアクセスします。
`CameraDevice` クラスの `takePicture()` メソッドや `CameraDevice` クラスの `startRecord()` メソッドに下記のパスを指定します。

[SD カードのパス]/Android/data/[パッケージ名]/[ファイル名]

カメラ制御の注意事項

Moverio Basic Function SDK を使用したカメラ制御は、単一のアプリでのみ利用することができます。別のアプリでカメラ制御を利用する場合は、必ずカメラ制御を利用しているアプリでカメラ制御の利用を停止してから、利用してください。

オーディオ制御の概要

Moverio は CTIA 規格に準拠したイヤホンで音の入出力ができます。Moverio Basic Function SDK ではイヤホンの音量を調整することができます。

関連する情報を参照してください。

- [イヤホンの音量の調整](#)
- [オーディオゲインの調整](#)
- [ヘッドセット音声出力モードの設定](#)
- [オーディオ制御機能の機種対応表](#)
- [API リファレンス](#)
- [サンプルコード](#)

イヤホンの音量の調整

周辺環境に合わせてイヤホンの音量を増減させることができます。

イヤホンの音量の調整には、最初に Moverio のオーディオとの通信を確立します。これを行うには、[AudioManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出してください。

※Moverio Basic Function SDK の [AudioManager](#) クラスの利用の際に、Android 標準の AudioManager クラスとの名前衝突を回避するために、com.epson.moverio.hardware.audio.AudioManager のインポートをする、もしくは、インスタンス生成時のクラス名を com.epson.moverio.hardware.audio.AudioManager としてください。

例) com.epson.moverio.hardware.audio.AudioManager のインポート

```
import com.epson.moverio.hardware.audio.AudioManager;
```

例) com.epson.moverio.hardware.audio.AudioManager のインスタンス生成

```
public class AudioActivity extends Activity {  
    private com.epson.moverio.hardware.audio.AudioManager mAudioManager = null;
```

そして、イヤホンの音量を調整するには、[setVolume\(\)](#) メソッドを使用してイヤホンの音量を調整してください。調整できるイヤホンの音量の範囲は、[イヤホンの音量の範囲](#)を参照してください。

最後に、イヤホンの音量の調整を完了した場合は、必ず Moverio のオーディオとの通信を解除してください。解除には、[close\(\)](#) メソッドを使用してください。

```
private AudioManager mAudioManager = new AudioManager(context);
try {
    mAudioManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
mAudioManager.setVolume(12);
mAudioManager.close();
```

オーディオゲインの調整

注意：オーディオゲイン調整の機能を利用することで、大音量になる場合があります。
機能を利用する際には、必ず製品マニュアルの「安全にお使いいただくために」を確認してください。

BT-45C のビルトインスピーカーは、出力音量を増幅させることができます。

オーディオゲインの調整機能を利用するためには、AndroidManifest.xml にオーディオゲイン調整用のカスタムパーミッションを宣言する必要があります。カスタムパーミッションは、build.gradle で宣言されている applicationId と ".permission.AUDIO_GAIN" を連結した文字列となります。\${applicationId} とすることで、build.gradle で定義された applicationId を参照できます。

注意：BasicFunctionSDK Version 1.2.0 を利用して、Android 11 以降のスマートフォンでオーディオゲイン調整の機能を利用する場合は、AndroidManifest.xml に permission-group を "
com.epson.moverio.permission-group.MOVERIO" として定義する必要があります。

注意：BasicFunctionSDK Version 1.2.1 以降では、オーディオゲイン調整の機能を利用する場合に、AndroidManifest.xml に permission-group を指定する必要はありません。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.moverio.app.moveriosdksample2">
    :
    :
    <!-- Custom permission of Audio gain control -->
    <uses-permission android:name="${applicationId}.permission.AUDIO_GAIN" />

    <!-- The following required Android 11 or later for BasicFunctionSDK Version 1.2.0 only. -->
    <permission-group
        android:name="com.epson.moverio.permission-group.MOVERIO"
        android:label="@string/perm_label_audio_gain_control"
        android:description="@string/perm_description_audio_gain_control" />
    :
    :
```



```
</manifest>
```

オーディオゲインの調整には、最初に Moverio のオーディオとの通信を確立します。これを行うには、[AudioManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出してください。

※ Moverio Basic Function SDK の [AudioManager](#) クラスの利用の際に、Android 標準の AudioManager クラスとの名前衝突を回避するために、com.epson.moverio.hardware.audio.AudioManager のインポートをする、もしくは、インスタンス生成時のクラス名を com.epson.moverio.hardware.audio.AudioManager としてください。

例) com.epson.moverio.hardware.audio.AudioManager のインポート

```
import com.epson.moverio.hardware.audio.AudioManager;
```

例) com.epson.moverio.hardware.audio.AudioManager のインスタンス生成

```
public class AudioActivity extends Activity {  
    private com.epson.moverio.hardware.audio.AudioManager mAudioManager = null;
```

そして、オーディオゲインを調整するには、[setGainStep\(\)](#) メソッドを使用してイヤホンの音量を調整してください。調整できるオーディオゲインの範囲は、[オーディオゲインの範囲](#)を参照してください。

最後に、オーディオゲインの調整を完了した場合は、必ず Moverio のオーディオとの通信を解除してください。解除には、[close\(\)](#) メソッドを使用してください。

```
public class AudioActivity extends Activity implements  
    ActivityCompat.OnRequestPermissionsResultCallback {  
  
    private AudioManager mAudioManager = null;  
    private PermissionHelper mPermissionHelper = null;  
    private Button mButton = null;  
  
    @Override  
    public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        mAudioManager = new AudioManager(this);  
        try {  
            mAudioManager.open();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
  
        :  
        mButton_gainStepUp.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                mAudioManager.setGainStep(3);
```

```

    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    mAudioManager.close();
    mAudioManager.release();
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
}

```

ヘッドセット音声出力モードの設定

注意：BT-45C のオーディオ機能を利用する場合は、USB デバイスへのアクセス許可を 2 回行う必要があります。

注意：ヘッドセット音声出力モードの設定を完了した場合、Moverio のオーディオとの通信が自動的に解除されます。

再度、MoverioBasicFunctionSDK で提供されるオーディオ機能を利用する場合は、必ず `open()` メソッドを呼び出して下さい。

BT-45C は、ヘッドセット音声出力モードを設定することができます。

ヘッドセット音声出力モードの設定機能を利用するためには、最初に Moverio のオーディオとの通信を確立します。これを行うには、`AudioManager` クラスのインスタンスを作成して、`open()` メソッドを呼び出してください。

※Moverio Basic Function SDK の `AudioManager` クラスの利用の際に、Android 標準の `AudioManager` クラスとの名前衝突を回避するために、`com.epson.moverio.hardware.audio.AudioManager` のインポートをする、もしくは、インスタンス生成時のクラス名を `com.epson.moverio.hardware.audio.AudioManager` としてください。

例) `com.epson.moverio.hardware.audio.AudioManager` のインポート

```
import com.epson.moverio.hardware.audio.AudioManager;
```

例) `com.epson.moverio.hardware.audio.AudioManager` のインスタンス生成

```
public class AudioActivity extends Activity {
    private com.epson.moverio.hardware.audio.AudioManager mAudioManager = null;
```

そして、ヘッドセット音声出力モードを設定するには、`setDeviceMode()` メソッドを使用してください。設定できるヘッドセット音声出力モードは、以下を参照してください。

デバイスモード	説明
DEVICE_MODE_BUILTIN_AUDIO	ビルトインスピーカー
DEVICE_MODE_AUDIO_JACK	オーディオジャック

最後に、ヘッドセット音声出力モードの設定を完了した場合は、Moverio のオーディオとの通信が自動的に解除されます。再度、MoverioBasicFunctionSDK で提供されるオーディオ機能を利用する場合は、必ず `open()` メソッドを呼び出してください。

```
public class AudioActivity extends Activity implements
    ActivityCompat.OnRequestPermissionsResultCallback {

    private AudioManager mAudioManager = null;
    private PermissionHelper mPermissionHelper = null;
    private Button mButton = null;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mAudioManager = new AudioManager(this);
        try {
            mAudioManager.open();
        } catch (IOException e) {
            e.printStackTrace();
        }

        :
        mButton_deviceMode.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mAudioManager.setDeviceMode(AudioManager.DEVICE_MODE_BUILTIN_AUDIO);
            }
        })
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mAudioManager.close();
        mAudioManager.release();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        mPermissionHelper.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

}

}

デバイス管理の概要

Moverio は、ホスト機器との USB 接続/抜去やディスプレイの表示開始、各種デバイスの温度異常などの Moverio を利用する上で重要なシステム状態の変化を知ることができます。また、Moverio の機器固有の情報を使用できます。Moverio Basic Function SDK ではシステムの状態の変化を検知することや機器固有の情報を取得することができます。システム状態の変化を検知することで、アプリケーションで Moverio が接続されたことを検知して機能を実行することやデバイスが高温になったことを検知してユーザーへ通知することができます。そして、機器固有の情報を使用することで、アプリケーションで特定の Moverio のみを動作させる機器認証の仕組みを導入することができます。

関連する情報を参照してください。

- [システム状態の変化の検知](#)
- [機器固有の情報の取得](#)
- [デバイス管理機能の機種対応表](#)
- [API リファレンス](#)
- [サンプルコード](#)

システム状態の変化の検知

Moverio Basic Function SDK では、USB 接続/抜去やディスプレイの表示開始、各種デバイスの温度異常などの Moverio を利用する上で重要なシステム状態の変化を検知することができます。システム状態の変化を検知することで、アプリケーションで Moverio が接続されたことを検知して機能を実行することやデバイスが高温になったことを検知してユーザーへ通知することができます。

システム状態の変化の検知には、`com.epson.moverio.system.HeadsetStateCallback` のインポートしてください。

```
import com.epson.moverio.system.HeadsetStateCallback;
```

この機能を利用するクラスに `HeadsetStateCallback` インターフェースを実装し、`registerHeadsetStateCallback` を実行してください。ホスト機器との USB 接続を検知した場合は `onHeadsetAttached()` が呼び出され、USB 抜去を検知した場合は `onHeadsetDetached()` が呼び出されます。また、ディスプレイの表示開始、各種デバイスの温度異常のシステム状態の変化を検知するためには、`DeviceManager` クラスのインスタンスを作成して、`open()` メソッドの実行が完了している必要があります。`DeviceManager#open()` メソッドの実行が完了する前に、Moverio のディスプレイの表示が開始された場合や、各種デバイスの温度異常が発生してしまった場合は、システム状態の変化は検知することができません。最後に、`DeviceManager#open()` メソッドを実行したシステム状態の変化の検知を終了させる場合は、必ず Moverio との通信を解除してください。解除には、`close()` メソッドと `release()` メソッドと `unregisterHeadsetStateCallback()` を使用してください。

```

public class HeadsetStateActivity extends Activity implements HeadsetStateCallback {
    private DeviceManager mDeviceManager = null;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mDeviceManager = new DeviceManager(this);
        mDeviceManager.registerHeadsetStateCallback(this);
    }

    @Override
    public void onDestroy(){
        super.onDestroy();
        mDeviceManager.unregisterHeadsetStateCallback(this);
    }

    @Override
    public void onHeadsetAttached() {
        // Get notified when a headset connection is detected.
        try {
            mDeviceManager.open();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onHeadsetDetached() {
        // Get notified when a headset disconnection is detected.
        mDeviceManager.close();
        mDeviceManager.release();
    }

    @Override
    public void onHeadsetDisplaying() {
        // Get notified when a headset displaying started.
    }

    @Override
    public void onHeadsetTemperatureError() {
        // Get notified when a headset temperature error happened.
    }
}

```

機器固有の情報の取得

Moverio はモデル情報やヘッドセットシリアル番号などの機器固有の情報を使用できます。機器固有の情報を使用することで、アプリケーションで特定の Moverio のみを動作させる機器認証の仕組みを導入することができます。

機器固有の情報の取得には、最初に Moverio との通信を確立します。これを行うには、[DeviceManager](#) クラスのインスタンスを作成して、[open\(\)](#) メソッドを呼び出してください。

次に、専用の API を使用することで機器固有の情報を取得することができます。モデル情報の取得には、[getHeadsetProductName\(\)](#) メソッドを使用します。ヘッドセットシリアル番号の取得には、まず、[isHeadsetSerialNumberAcquisitionSupported\(\)](#) メソッドで接続されている Moverio がヘッドセットシリアル番号の取得ができることを確認し、[getHeadsetSerialNumber\(\)](#) メソッドを使用します。

最後に、機器固有の情報の取得を完了した場合は、必ず Moverio との通信を解除してください。解除には、[close\(\)](#) メソッドと [release\(\)](#) メソッドを使用してください。

```
private DeviceManager mDeviceManager = new DeviceManager(context);
mDeviceManager = new DeviceManager(this);
try {
    mDeviceManager.open();
} catch (IOException e) {
    e.printStackTrace();
}
// Model information
Log.v("sample", mDeviceManager.getHeadsetProductName());
// Headset serial information
if (mDeviceManager.isHeadsetSerialNumberAcquisitionSupported()){
    Log.v("sample", mDeviceManager.getHeadsetSerialNumber());
}
mDeviceManager.close();
mDeviceManager.release();
```

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

BT-35E などの USB 接続型 Moverio のアプリケーション開発では、USB 接続型 Moverio と Android スマートフォンを USB 接続し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 Android スマートフォンは 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 Android スマートフォンに adb 接続することで USB 接続型 Moverio と Android スマートフォンを 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 Android スマートフォンを同じネットワークに接続します。

2. パソコンと Android スマートフォンを USB 接続します。

※以降、パソコンに接続されている Android スマートフォンは 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、Android スマートフォン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. Android スマートフォンの IP アドレスを確認します。

いくつかの Android スマートフォンにおいて IP アドレスは設定画面から確認できます。IP アドレスが表示される画面までの手順は Android スマートフォンによって異なります。詳しくはお手持ちの Android スマートフォンの操作方法を確認ください。

5. パソコンにおいて下記コマンドを実行し、Android スマートフォン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

6. パソコンと Android スマートフォンの USB 接続を解除します。

7. パソコンにおいて下記コマンドを実行し、パソコンに Android スマートフォンが 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 Android スマートフォン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。

Kotlin から Moverio Basic Function SDK を利用する

Kotlin から Moverio Basic Function SDK を利用する手順について以下に記載します。

- [Moverio Basic Function SDK の組み込み](#)
- [Moverio Basic Function SDK API の実行例](#)

Moverio Basic Function SDK の組み込み

Kotlin が開発言語として選択されているプロジェクトに対して Moverio Basic Function SDK の組み込みを実施してください。
Moverio Basic Function SDK の組み込み方法は以下を確認してください。

➤ [Android アプリ開発の概要](#)

Moverio Basic Function SDK API の実行例

Kotlin からの Moverio Basic Function SDK API の実行例を記載します。

ディスプレイ制御

明るさの取得/設定の実行例を記載します。

ディスプレイ制御の詳細は以下を確認してください。

➤ [ディスプレイ制御の概要](#)

```
val displayManager = DisplayManager(context)
try {
    displayManager.open()
} catch (e: IOException) {
    e.printStackTrace()
}

val brightness = displayManager.brightness // getBrightness
displayManager.brightness = brightness + 1 // setBrightness

displayManager.close()
```

センサー制御

加速度センサーのデータ取得の実行例を記載します。

センサー制御の詳細は以下を確認してください。

➤ [センサー制御の概要](#)

```
class SensorActivity : Activity(), SensorDataListener {

    private var sensorManager: SensorManager? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        sensorManager = SensorManager(this)
    }

    override fun onResume() {
        super.onResume()
        try {
            sensorManager?.open(SensorManager.TYPE_ACCELEROMETER, this)
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }

    override fun onPause() {
        super.onPause()
        sensorManager?.close(this)
    }

    override fun onSensorDataChanged(SensorData data) {
        // Do something with this sensor value.
    }
}
```

カメラ制御

プレビュー取得の実行例を記載します。

カメラ制御の詳細は以下を確認してください。

➤ [カメラ制御の概要](#)

```
class CameraActivity : Activity() {

    private var cameraManager: CameraManager? = null

    private var cameraDevice: CameraDevice? = null

    private var surfaceView: SurfaceView? = null

    private var captureStateCallback = object : CaptureStateCallback {
        override fun onCaptureStarted() {
            cameraDevice?.startPreview()
        }
        :
        省略
        :
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.camera)

        surfaceView = findViewById(R.id.surfaceView)
        cameraManager = CameraManager(this)
        try {
            cameraManager?.open(captureStateCallback, null, surfaceView?.holder)
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }

    override fun onResume() {
        super.onResume()
        cameraDevice?.startCapture()
    }

    override fun onPause() {
        super.onPause()
        cameraDevice?.stopPreview()
    }
}
```

```
        cameraDevice?.stopCapture()
    }

    override fun onDestroy() {
        super.onDestroy()
        cameraManager?.close(cameraDevice)
    }
}
```

Android のマルチディスプレイの活用

Android は、プラットフォームで複数のディスプレイへの画面出力をサポートしています。方法は 2 通りあり、“Presentaion クラスを使う方法”と“Android8 以降のマルチディスプレイの機能を利用する方法”があります。“Presentaion クラスを使う方法”は、API level 17 以降の対応端末で利用可能です。“Android8 以降のマルチディスプレイの機能を利用する方法”は、Android8 以降の対応端末で利用可能です。“Android8 以降のマルチディスプレイの機能を利用する方法”は、既存のアプリケーションを改変することなく、Moverio へ画面出力することができます。ただし、アプリケーションへの入力操作などの対応は、機能を利用する端末に依存します。これらの Android プラットフォームの機能を活用することで、Moverio への画面出力と端末の画面でそれぞれ構成することができます。

注意：“Android8 以降のマルチディスプレイの機能を利用する方法”では、既存のアプリケーションの Activity が、マルチウィンドウモードに対応している必要があります。

関連する情報を参照してください。

- [Presentaion クラスを使う方法](#)（外部サイト）
- [Android8 以降のマルチディスプレイの機能を利用する方法](#)（外部サイト）
- サンプルコード

Windows アプリ開発の概要

Moverio 向けの Windows 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以下に記載します。

- [Windows SDK の導入](#)
- [アプリケーションの作成](#)
- [Visual Studio ドキュメント](#) (外部サイト)

Windows SDK の導入

以下の項目では Windows10 を搭載したパソコンに Windows SDK を導入する方法について記載します。

Visual Studio 2019 の入手

Visual Studio 2019 を下記のサイトからダウンロードします。(2020 年 7 月時点の最新バージョンは、Visual Studio 2019)
<https://visualstudio.microsoft.com/ja/vs/>

Visual Studio 2019 のインストール

Visual Studio Installer を起動し、指示に従い Visual Studio 2019 をインストールします。

ワークロードの追加

Visual Studio Installer を起動し、ワークロードの画面で「Universal Windows Platform development」と「.NET desktop development」と「Desktop development with C++」を追加してインストールします。下記のサイトで詳細な手順を確認してください。

<https://docs.microsoft.com/ja-jp/visualstudio/install/modify-visual-studio?view=vs-2019>

困ったときは

Microsoft 社の Visual Studio ドキュメントを確認してください。

<https://docs.microsoft.com/ja-jp/visualstudio/?view=vs-2019>

アプリケーションの作成

本ドキュメントにおけるアプリケーションの作成は、デスクトップアプリの作成について記載します。お客様の開発するアプリケーションに合わせてワークロードを選択してください。

デスクトップアプリの作成

下記のサイトで詳細な手順を確認してください。

<https://docs.microsoft.com/ja-jp/windows/win32/>

ディスプレイ制御の概要(for Windows)

Moverio はシースルー型（透過型）のディスプレイを搭載しています。Windows デスクトップアプリでのディスプレイ制御は、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンドを送信することで制御することができます。専用のコマンドを使用することで、ディスプレイの明るさや 2D/3D 表示モードの切り替えなどを制御することができます。ディスプレイの明るさは専用のコマンドからの調整と周辺環境の照度に合わせた自動調整の方法を使用することができます。また、Side-by-Side 方式の 3D コンテンツの表示に対応することが可能です。

関連する情報を参照してください。

- [ディスプレイの明るさの調整](#)
- [2D/3D 表示モードの切り替え](#)
- [ディスプレイの表示/非表示の設定](#)
- [ディスプレイの輻輳の調整](#)
- [ディスプレイの自動スリープ設定](#)
- [ディスプレイのユーザースリープ設定](#)
- [シースルーを活かした映像の作り方](#)
- [ディスプレイ制御機能の機種対応表](#)
- [Windows 用コマンドリファレンス](#) ※別紙参照
- [サンプルコード](#)
- [System.IO.Ports.SerialPort クラス](#) (外部サイト)

ディスプレイの明るさの調整

Moverio はシースルー型のディスプレイを搭載しており、周辺環境の明るさの影響により表示映像の見やすさが変化します。周辺環境が明るい場合はディスプレイの明るさを強くし、周辺環境が暗い場合はディスプレイの明るさを弱くすることで表示映像を見やすくすることができます。

ディスプレイの明るさを調整するには、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド“setbright **xx**” (xx は 0~20 を指定)を実行してください。ディスプレイの明るさは、0~20 の 21 段階で調整可能です。ディスプレイの明るさ調整モードを自動モードに設定するには、専用のコマンド“setbright 50”を実行してください。ディスプレイの明るさ調整モードが自動モードの場合に、専用コマンド“setbright **xx**” (xx は 0~20 を指定)を実行した場合は、明るさ調整モードが手動モードに切り替わります。

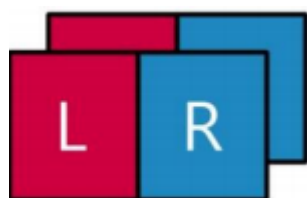
また、現在のディスプレイの明るさを取得するには、専用のコマンド“getbright”を実行してください。ディスプレイの明るさ調整モードが手動モードの場合はコマンドが 0~20 を返却し、自動モードの場合はコマンドが 50 を返却します。

調整できるディスプレイの明るさの範囲は、[ディスプレイの明るさの範囲](#)を参照してください。

2D/3D 表示モードの切り替え

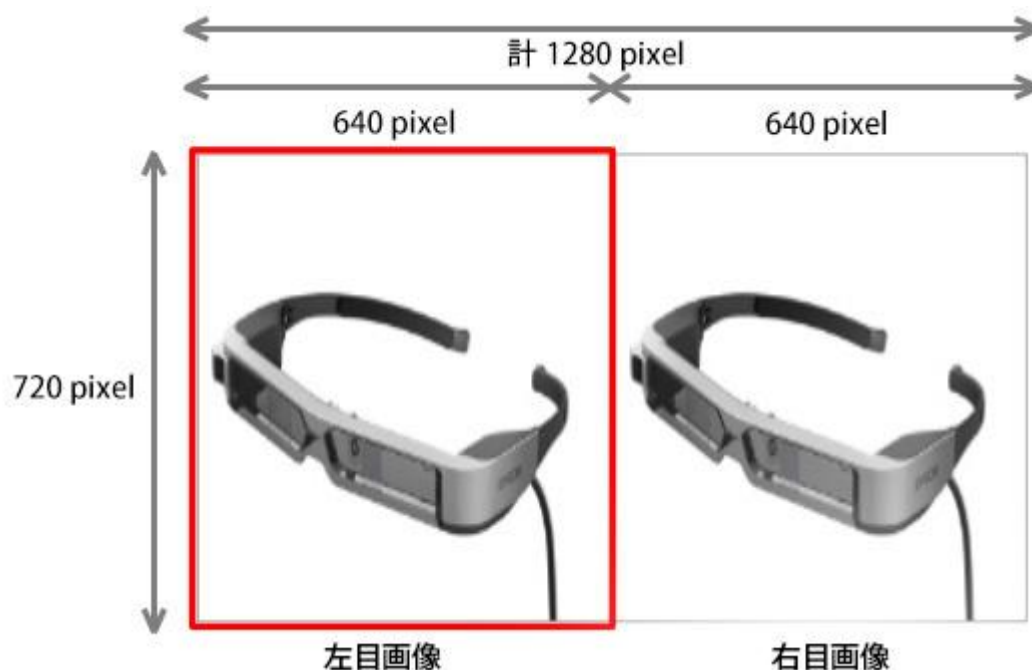
Moverio のディスプレイは、Side-by-Side 方式の 3D コンテンツの表示に対応しています。Moverio で表示する映像が Side-by-Side 方式の 3D コンテンツの場合でアプリから制御する場合は、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド "set2d3d 0" (2D 表示モード)、または、専用のコマンド "set2d3d 1" (3D 表示モード) を実行してください。System.IO.Ports.SerialPort の使い方は、Microsoft 社の [SerialPort クラスのドキュメント \(外部サイト\)](#) を参照してください。

Side-by-Side 方式とは一つの画面に左右の画像を並べて格納する方式です。



Side-by-Side 方式の映像を作成する場合は、映像の左半分と右半分に、それぞれ左目用映像と右目用映像を配置してください。

例えば、1280x720 (HD サイズ) の Side-by-Side 方式の映像を作成する場合は、下図のように左目用映像に 640x720 の映像を、右目用映像に 640x720 の映像を配置することで Side-by-Side 方式の映像とすることができます。



ディスプレイの 2D/3D 表示モードの切り替えには、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド“set2d3d 0”(2D 表示モード)、または、専用のコマンド“set2d3d 1”(3D 表示モード)を実行してください。

また、現在のディスプレイの 2D/3D 表示モードを取得するには、専用のコマンド“get2d3d”を実行してください。ディスプレイの 2D/3D 表示モードが 2D 表示モードの場合はコマンドが 0 を返却し、3D 表示モードの場合はコマンドが 1 を返却します。

ディスプレイの表示/非表示の設定

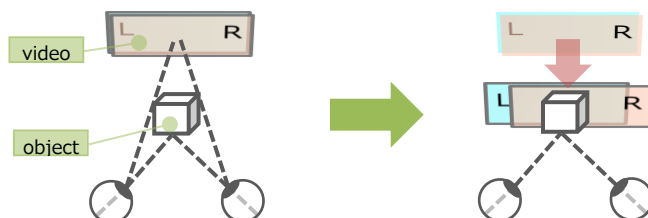
Moverio に表示されている映像を一時的に非表示にしたい場合は、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド“setmute 0”(ディスプレイ点灯)、または、専用のコマンド“setmute 1”(ディスプレイ消灯)を実行してください。System.IO.Ports.SerialPort の使い方は、Microsoft 社の [SerialPort クラスのドキュメント](#) (外部サイト) を参照してください。



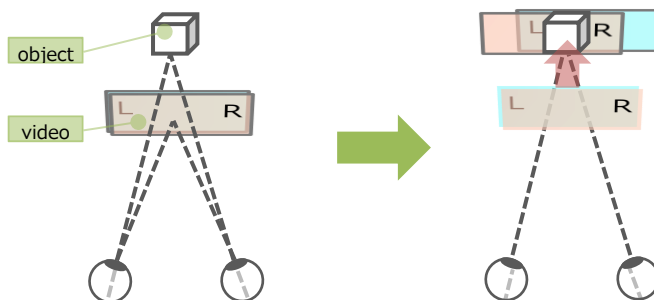
ディスプレイの輻輳の調整

Moverio のディスプレイの表示距離を専用のコマンドで調整することができます。専用のコマンドは左右の表示映像を水平方向にシフトする機能があります。水平方向のシフト量に応じてディスプレイの輻輳を調整できます。

例えば、実際のオブジェクトがディスプレイの表示距離よりも近い位置にある場合は、下図のように左右の映像を内側にシフトします。そうすることで、ディスプレイの仮想的な表示距離を近くにすることができます。



その他に、実際のオブジェクトがディスプレイの表示距離よりも遠い位置にある場合は、下図のように左右の映像を外側にシフトします。そうすることで、ディスプレイの仮想的な表示距離を遠くにすることができます。



ただし、この機能を利用することで左右の映像をシフトさせるため、左右の映像の端が途切れてしまいます。そのため、アプリケーションで映像のシフト量に応じた画面構成の配慮が必要になります。ディスプレイの仮想的な表示距離と映像のシフト量の関係性は、[ディスプレイの仮想的な表示距離の調整の範囲と水平方向のシフト量](#)を参照してください。

ディスプレイの仮想的な表示距離の調整には、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド "setdisplaydistance **xx**" (xx は画素シフト量を -32~256 で指定) を実行してください。System.IO.Ports.SerialPort の使い方は、Microsoft 社の [SerialPort クラスのドキュメント](#) (外部サイト) を参照してください。

専用コマンドで指定可能な画素シフト量は、[ディスプレイの仮想的な表示距離の調整の範囲と水平方向のシフト量](#)を参照してください。

また、現在の画素シフト量を取得するには、専用のコマンド "getdisplaydistance" を実行してください。

ディスプレイの自動スリープ設定

Moverio のディスプレイの自動スリープ設定を専用のコマンドで設定することができます。

ディスプレイの自動スリープ設定を有効化した場合は、Moverio を脱着して机上などに置くと、自動的にディスプレイを消灯します。また、Moverio を装着すると自動的にディスプレイを表示します。この機能を有効にすることで、Moverio を装着していない場合の消費電力を低減させることができます。

ディスプレイの自動スリープ設定には、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド "enableautosleep 0" (自動スリープ OFF)、または、専用のコマンド "enableautosleep 1" (自動スリープ ON) を実行してください。

また、現在のディスプレイの仮想的な表示距離の調整値を取得するには、専用のコマンド "getautosleep" を実行してください。

ディスプレイのユーザースリープ設定

Moverio のディスプレイのユーザースリープ設定を専用のコマンドで設定することができます。

ディスプレイのユーザースリープ設定を有効化した場合は、Moverio の側面を軽くタップするとディスプレイを消灯します。また、再度 Moverio の側面を軽くタップするとディスプレイを表示します。この機能は、Moverio の装着者が前面の映像を即座に消したい時などに利用できます。

ディスプレイのユーザースリープ設定には、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド "enableusersleep 0" (自動スリープ OFF)、または、専用のコマンド "enableusersleep 1" (自動スリープ ON) を実行してください。System.IO.Ports.SerialPort の使い方は、Microsoft 社の [SerialPort クラスのドキュメント](#) (外部サイト) を参照してください。

また、現在のディスプレイの仮想的な表示距離の調整値を取得するには、専用のコマンド "getusersleep" を実行してください。

シースルーを活かした映像の作り方

Moverio はシースルー型のディスプレイを搭載しており、Moverio の装着者は実空間で見えている景色を見ながら表示される映像を重ね合わせてみることができます。実空間で見えている景色に Moverio で表示したいオブジェクトの映像を重ね合わせて見る場合は、表示したいオブジェクトの背景を黒にすることで、シースルーを活かした映像を作ることができます。



センサー制御の概要(for Windows)

Moverio は動き、向き、環境を検知する様々なセンサーをヘッドセットに搭載しています。Windows デスクトップアプリでのセンサー制御は、標準のセンサー制御 API である Sensor API を使用します。センサーデータを使用することで、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。

関連する情報を参照してください。

- [センサーの種類](#)
- [センサーの軸](#)
- [Sensor API について \(外部サイト\)](#)
- [センサー制御機能の機種対応表](#)
- [API リファレンス \(外部サイト\)](#)
- [サンプルコード \(外部サイト\)](#)

センサーの種類

Moverio は様々な種類のセンサーを搭載しています。センサーの中には、ハードウェアベースのセンサーとソフトウェアベースのセンサーがあります。ハードウェアベースのセンサーには、加速度、地磁気、角速度、環境照度などがあります。ソフトウェアベースのセンサーは、単体もしくは複数のハードウェアベースのセンサーの出力値を使用した仮想のセンサーです。ソフトウェアベースのセンサーには、回転ベクトルや重力などの推測が可能なものがあります。

センサー	種類	説明	使用例
SENSOR_TYPE_ACCELEROMETER_3D	ハードウェア	重力を含む Moverio の加速度を 3 軸 (x、y、z) の加速度[G]で測定します。	モーション検知 (傾きなど)
SENSOR_TYPE_COMPASS_3D	ハードウェア	周囲の地磁気を 3 軸 (x、y、z) の地磁気[mG]で測定します。	方位検知
SENSOR_TYPE_GYROMETER_3D	ハードウェア	Moverio の角速度を 3 軸 (x、y、z) の角速度[deg/s]で測定します。	モーション検知 (回転など)
SENSOR_TYPE_AMBIENT_LIGHT	ハードウェア	周囲の照度[lx]を測定します。	周囲の照度に合わせたディスプレイの明るさ調整
GUID_SensorType_GravityVector	ソフトウェア	重力を 3 軸 (x、y、z) の加速度[G]で測定します。	モーション検知 (傾きなど)
GUID_SensorType_LinearAccelerometer	ソフトウェア	重力を除いた 3 軸 (x、y、z) の加速度[G]で測定します。	タップ、歩行の検出などへの応用
GUID_SensorType_RelativeOrientation	ソフトウェア	Moverio の向きを、地磁気を使用しないクォータニオンで測定します。	ヘッドトラッキングなど
SENSOR_TYPE_AGGREGATED	ソフトウェア	Moverio の向きをクォータニオンで測定しま	ヘッドトラッキングなど

_DEVICE_ORIENTATION		す。	
SENSOR_TYPE_CUSTOM (0)*1	ハードウェア	重力を含む Moverio の加速度を未校正の 3 軸 (x、y、z) の加速度[G]と推定バイアス補償の 3 軸 (x_bias, y_bias, z_bias) で測定します。	モーション検知 (傾きなど)、ヘッドトラッキングへの応用
SENSOR_TYPE_CUSTOM (1)*1	ハードウェア	Moverio の角速度を未校正の 3 軸 (x、y、z) の角速度[deg/s]と推定ドリフトの 3 軸 (x_bias, y_bias, z_bias) で測定します。	モーション検知 (回転など)、ヘッドトラッキングへの応用
SENSOR_TYPE_CUSTOM (2)*1	ハードウェア	周囲の地磁気を未校正の 3 軸 (x、y、z) の地磁気 [mG]と校正情報の 3 軸 (x_bias, y_bias, z_bias) で測定します。	方位検知、ヘッドトラッキングへの応用
SENSOR_TYPE_CUSTOM (3)*1	ソフトウェア	Moverio の静止/動作を検知します。	ヘッドセット着脱検知など
SENSOR_TYPE_CUSTOM (4)*1	ソフトウェア	ヘッドセットのタップを検知します。	ヘッドセットタップ検知

*1 SENSOR_TYPE_CUSTOM の場合は、SENSOR_DATA_TYPE_CUSTOM_USAGE の数値と上表の括弧内の数値で照らし合わせて判別。

下記のサイトで詳細な情報を確認してください。

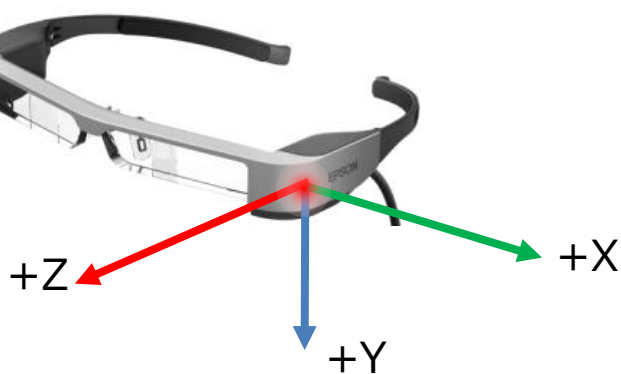
<https://docs.microsoft.com/ja-jp/windows/win32/sensorsapi/sensor-categories--types--and-datafields>

<https://docs.microsoft.com/ja-jp/windows-hardware/drivers/sensors/sensor-types>

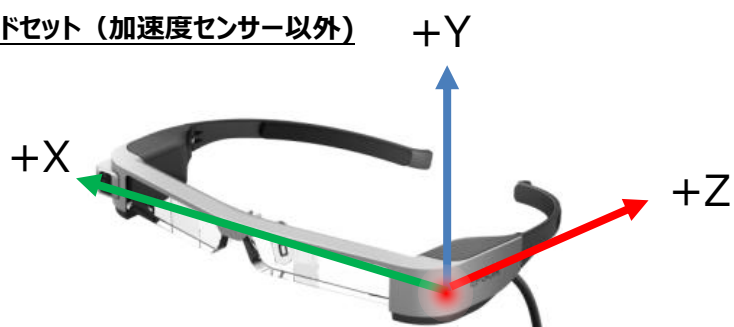
センサーの軸

Moverio のセンサーは、加速度センサーとそれ以外のセンサーで座標系が異なります。Moverio を装着した場合、加速度センサーの X 軸は左方向を指し、Y 軸は下方向を指し、Z 軸は装着者の視線方向を指します。加速度センサー以外の 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。

ヘッドセット（加速度センサーの場合）



ヘッドセット（加速度センサー以外）



カメラ制御の概要(for Windows)

Moverio は通常の撮影やマーカ認識などの用途で使うためにカメラをヘッドセットに搭載しています。Windows デスクトップアプリでのカメラ制御は、Windows 標準のカメラ制御 API である Microsoft Media Foundation を使用します。この API では、動画撮影やプレビューの表示などができます。

関連する情報を参照してください。

- [Microsoft Media Foundation について](#) (外部サイト)
- [カメラ制御機能の機種対応表](#)
- [API リファレンス](#) (外部サイト)
- [サンプルコード](#) (外部サイト)

オーディオ制御の概要(for Windows)

Moverio は音源の聴取や音声通話などの用途で使うために CTIA 準拠のイヤホン・マイクを接続できます。Windows デスクトップアプリでのオーディオ制御するためには、Windows 標準のオーディオ制御 API である Core Audio APIs を使用します。この API では、音源のイヤホン出力や音声のマイク入力、イヤホン・マイクの各種パラメータ設定などができます。

関連する情報を参照してください。

- [Core Audio APIs について](#) (外部サイト)
- [音源のイヤホン出力](#)
- [音声のマイク入力](#)
- [各種パラメータ設定](#)
- [ミュート設定](#)
- [オーディオ制御機能の機種対応表](#)
- [API リファレンス](#) (外部サイト)

音源のイヤホン出力

Moverio は、Core Audio APIs を利用して接続されたイヤホンから音源を聴取することができます。下記のサイトで詳細な情報を確認してください。

<https://docs.microsoft.com/ja-jp/windows/win32/coreaudio/rendershareeventdriven>

音声のマイク入力

Moverio は、Core Audio APIs を利用して接続されたマイクから音声を入力することができます。下記のサイトで詳細な情報を確認してください。

<https://docs.microsoft.com/ja-jp/windows/win32/coreaudio/captureshareeventdriven>

各種パラメータ設定

Moverio は、Core Audio APIs を利用して接続されたイヤホン・マイクの各種パラメータの設定が可能です。下記のサイトで詳細な情報を確認してください。

<https://docs.microsoft.com/ja-jp/windows/win32/coreaudio/programming-guide>

ミュート設定

Moverio は、Core Audio APIs を利用してイヤホン出力やマイク入力を一時的に停止（ミュート設定）することが可能です。下記のサイトで詳細な情報を確認してください。

<https://docs.microsoft.com/ja-jp/windows/win32/api/audiopolicy/nf-audiopolicy-iaudiosessionevents-onsimplevolumechanged>

デバイス管理の概要(for Windows)

Moverio は、ホスト機器とのディスプレイの表示開始、各種デバイスの温度異常などの Moverio を利用する上で重要なシステム状態の変化を監視することができます。また、Moverio の機器固有の情報を使用できます。システム状態の変化を監視することで、アプリケーションでデバイスが高温になったことを検知してユーザーへ通知することができます。そして、機器固有の情報を使用することで、アプリケーションで特定の Moverio のみを動作させる機器認証の仕組みを導入することができます。

関連する情報を参照してください。

- [Moverio のモデル判別](#)
- [システム状態の変化の監視](#)
- [機器固有の情報の取得](#)
- [デバイス管理機能の機種対応表](#) (リンク…「01_Moverio 機能の機種対応表」)
- [Windows 用コマンドリファレンス](#) ※別紙参照
- [System.IO.Ports.SerialPort クラス](#) (外部サイト)

Moverio のモデル判別

Moverio はモデルごとに異なる機能を提供しています。カメラ機能の有無や対応するセンサーの種類などモデルごとに異なります。そこで、Windows 端末に接続された Moverio のモデルは、USB CDC (Communication Device Class) 情報の Vendor ID と Product ID を用いて判別します。詳細は下表を参照してください。

USB Property	BT-35E/30E	BT-30C	BT-40	BT-45C
Vendor ID	VID_0483	VID_04B8	VID_04B8	VID_04B8
Product ID	PID_5750	PID_0C0C	PID_0D12	PID_0C0E

Windows 端末に接続された Moverio の USB CDC 情報を取得するためには、.NET API の ManagementClass クラスを使用します。このクラスのインスタンス生成時に "Win32_SerialPort" を指定し、ポートごとに "PNPDeviceID" の情報を取得し、上表と照らし合わせてモデルを判別します。

```
ManagementClass mcW32SerPort = new ManagementClass("Win32_SerialPort");
foreach (ManagementObject port in mcW32SerPort.GetInstances()) {
    string pnpDeviceId = (string)port.GetPropertyValue("PNPDeviceID");
    if (pnpDeviceId.Contains("VID_0483") && pnpDeviceId.Contains("PID_5750")) {
        Console.WriteLine("BT-35E/30E detected.");
    }
    else if (pnpDeviceId.Contains("VID_04B8") && pnpDeviceId.Contains("PID_0C0C")) {
        Console.WriteLine("BT-30C detected.");
    }
    else if (pnpDeviceId.Contains("VID_04B8") && pnpDeviceId.Contains("PID_0D12")) {
```

```

        Console.WriteLine("BT-40 detected.");
    }
    else if (pnpDeviceId.Contains("VID_04B8") && pnpDeviceId.Contains("PID_0C0E")) {
        Console.WriteLine("BT-45C detected.");
    }
    else {
        Console.WriteLine("Unknown device");
    }
}
}

```

下記のサイトで ManagementClass クラスの詳細な情報を確認してください。

<https://docs.microsoft.com/ja-jp/dotnet/api/system.management.managementclass?view=dotnet-plat-ext-3.1>

システム状態の変化の監視

Windows では、ディスプレイの表示開始、各種デバイスの温度異常などの Moverio を利用する上で重要なシステム状態の変化を監視することができます。システム状態の変化を監視することで、アプリケーションでデバイスが高温になったことを検知してユーザーへ通知することができます。

システム状態の変化を監視する場合は、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド "getsystemstat" (システム状態取得) を実行してください。System.IO.Ports.SerialPort の使い方は、Microsoft 社の [SerialPort クラスのドキュメント](#) (外部サイト) を参照してください。

専用のコマンドの実行結果は、下表の通りです。

戻り値	説明
0	電源 OFF
1	Moverio のシステムを初期化している状態
2	ホスト機器から映像出力がない状態
3	映像表示中
4	ディスプレイ消灯
5	リカバリー状態 ※ 静電気の発生などによりディスプレイ消灯が発生した場合の自動復帰機能の動作状態

機器固有の情報の取得

Moverio はヘッドセットシリアル番号の情報を使用できます。機器固有の情報を使用することで、アプリケーションで特定の Moverio のみを動作させる機器認証の仕組みを導入することができます。

機器固有の情報の取得には、Windows 標準の COM ポートアクセス API である System.IO.Ports.SerialPort を使用し、専用のコマンド "gethserial" (ヘッドセットシリアル番号取得) を実行してください。System.IO.Ports.SerialPort の使い方は、Microsoft 社の [SerialPort クラスのドキュメント](#) (外部サイト) を参照してください。

Appendix

ディスプレイの明るさの範囲

調整できるディスプレイの明るさの範囲は、下表を参照してください。

機種	明るさの範囲
BT-35E/30E	0 ~ 20
BT-30C	0 ~ 20
BT-40	0 ~ 20
BT-45C	0 ~ 20

ディスプレイの仮想的な表示距離の調整の範囲と水平方向のシフト量

調整できるディスプレイの仮想的な表示距離の範囲と水平方向のシフト量は、下表を参照してください。

下表の表示距離は参考値であり、精度を保証するものではありません。

※BT-35E/30E や BT-30C は、非対応となります。

ステップ	BT-40		BT-45C	
	画素シフト量	表示距離[m]※参考値	画素シフト量	表示距離[m]※参考値
0	256	0.76	256	0.76
1	248	0.79	248	0.79
2	240	0.81	240	0.81
3	232	0.83	232	0.83
4	224	0.85	224	0.85
5	216	0.88	216	0.88
6	208	0.91	208	0.91
7	200	0.94	200	0.94
8	192	0.97	192	0.97
9	184	1.00	184	1.00
10	176	1.03	176	1.03
11	168	1.07	168	1.07
12	160	1.11	160	1.11
13	152	1.16	152	1.16
14	144	1.20	144	1.20
15	136	1.26	136	1.26
16	128	1.31	128	1.31
17	120	1.37	120	1.37
18	112	1.44	112	1.44
19	104	1.51	104	1.51
20	96	1.60	96	1.60

21	88	1.69	88	1.69
22	80	1.79	80	1.79
23	72	1.91	72	1.91
24	64	2.04	64	2.04
25	56	2.19	56	2.19
26	48	2.37	48	2.37
27	40	2.58	40	2.58
28	32	2.83	32	2.83
29	24	3.13	24	3.13
30	16	3.50	16	3.50
31	8	3.98	8	3.98
32	0(default)	4.60	0(default)	4.60
33	-8	5.45	-8	5.45
34	-16	6.70	-16	6.70
35	-24	8.68	-24	8.68
36	-32	12.32	-32	12.32

イヤホンの音量の範囲

調整できるイヤホンの音量の範囲は、下表を参照してください。

機種	音量の範囲	
BT-35E/30E	0 ~ 15	
BT-30C	0 ~ 20	
BT-40	-	
BT-45C	-	

オーディオゲインの範囲

調整できるイヤホンの音量の範囲は、下表を参照してください。

機種	オーディオゲインの範囲	
BT-35E/30E	-	
BT-30C	-	
BT-40	-	
BT-45C	0 ~ +4 (default=0)	