

Developer's Guide

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.

Contents

- Overview of the Moverio software development
- Supported function by model
- Android application software development procedure
- Display control
- Sensor Control
- Camera control
- Audio control
- Device management
- Network debug
- Using Moverio Basic Function SDK from Kotlin
- About Android multi display
- Windows application development
- Windows display control
- Windows sensor control
- Windows camera control
- Windows audio control
- Windows device control

Summary of MOVERIO application software development

Android application software development, you need the Android Software Development Kit (Android SDK) which is provided by Google and Moverio dedicated software development kit (Moverio Basic Function SDK) which is provided by Epson. Using Moverio Basic Function SDK, your application software can use the display, the camera and the sensors which are equipped on the MOVERIO. Moverio Basic Function SDK is supported the BT35E or later model with compatible API, then your software can be work on newer MOVERIO. Using Unityplug in which si contained Moverio Basic Function SDK, you can use devices on the MOVERIO such as display, camera and sensors.

Windows application software can be develop with Windows standard development environment.

MOVERIO software development environment

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

※1 Unity 2018.4.0f1 or later is supported

Note : Regarding head set model

Headset model such as BT-40 can be connected by USB to Android smartphone and Windows PC.

However, some of MOVERIO function cannot be used by compatibility of Android smartphone and Windows PC.

It is not supported Linux PC.

Scope of application of Moverio Basic Function SDK

The Moverio Basic Function SDK (formerly Moverio SDK) will be upgraded to support additional models, so please make sure you use the latest version of the Moverio Basic Function SDK when using it.

The Moverio Basic Function SDK will keep compatibility with later model by updating of version. Please use latest version of the Moverio Basic Function SDK.

The Moverio Basic Function SDK is not compatible with prior to the BT-350. The Moverio Basic Function SDK is only applied to develop Android application software. When developing an Android application, it can coexist with the SDK provided for past models. However, you need to use dedicated SDK each model prior to the BT-350.

	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					✓	✓		
Moverio Basic Function SDK V1.1.0					✓	✓	✓	
Moverio Basic Function SDK V1.2.0					✓	✓	✓ Note	✓ Note
Moverio Basic Function SDK V1.2.1					✓	✓	✓ Note	✓ Note
Moverio Basic Function SDK V1.2.2					✓	✓	✓ Note	✓ Note

Note: The functions of Moverio's dedicated controller (model number: BO-IC400), which were provided in MoverioBasicFunctionSDK V1.1.0, are now provided in Moverio Controller Function SDK.

Supported function by Model

Each MOVERIO has slight different functionality, The Moverio Basic Function SDK supports those function by each model.

Display control

Function	BT-35E/30E	BT-30C	BT-40	BT-45C
Brightness adjustment (manual)	●	●	●	●
Brightness adjustment (auto)	●		●	●
Switch 2D/3D mode	●	●	●	●
Display on/off setting			●	●
Display pixel shift function			●	●
Display automatic sleep			●	●
Display manual sleep			●	●

Sensor control

Function	BT-35E/30E	BT-30C	BT-40	BT-45C
Accelerometer	●	●	●	●
Geomagnetic sensor	●	●	●	●
Gyro sensor	●	●	●	●
Illuminance sensor	●		●	●
Gravity sensor	●	●	●	●
Linear accelerometer	●	●	●	●
Rotation vector sensor	●	●	●	●
Geomagnetic sensor unused rotation vector sensor			●	●
Uncalibrated accelerometer			●	●
Uncalibrated geomagnetic sensor			●	●
Uncalibrated gyro sensor			●	●
Headset motion detection			●	●
Headset stillness detection			●	●
Headset tap detection			●	●

Camera control

Function	BT-35E/30E	BT-30C	BT-40	BT-45C
Preview display	●			●
Getting camera image	●			●

Shooting still image / movie	●			●
Setting capture format	●			●
Setting resolution/framerate	●			●
Exposure compensation mode setting	●			●
Manual exposure compensation step adjustment	●			●
Auto focus mode setting				●
Manual focus adjustment				●
Sharpness adjustment	●			
Brightness adjustment	●			●
Gain adjustment	●			●
White balance adjustment	●			●
Power line frequency setting (50Hz/60Hz)	●			●
Power line frequency setting (Disable)				●
Camera indicator mode setting				●

Audio control

Function	BT-35E/30E	BT-30C	BT-40	BT-45C
Adjusting the earphone volume	●	●		
Setting volume limit mode		●		
Adjusting audio gain				●
Setting audio device mode				●

Device information

Function	BT-35E/30E	BT-30C	BT-40	BT-45C
Headset system status	●	●	●	●
Headset serial number			●	●
Product name	●	●	●	●
Headset system version	●	●	●	●
Device tempreture			●	●

Android application software developmet procedure

Android SDK

Following is an explanation how to install Android SDK using Windows 10 PC as an example.

Download Android Studio

Android Studio will be downloaded from following Android developer's site.

Current version is Android Studio 4.0.1 (as of Aug. 2020)

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

Install Android Studio

Follow an installer instlaction and install Android Studio on your PC.

The later explanation is based on the SDK is installed in the following folder

C:\Users\<user name>\AppData\Local\Android\Sdk

Android Studio proxy settings

When developing an application in a network environment that requires proxy settings, set the Android Studio proxy settings. Please check the detailed procedure on the site below.

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

If you do not know the proxy settings, ask your network administrator how to connect to the external network using the proxy.

Acquiring/updating tools with Android SDK Manager

Use the Android SDK Manager for the tools required for application development. Please check the detailed procedure on the site below.

<https://developer.android.com/studio/intro/update#sdk-manager>

Note: If you want to build an app with targetSdkVersion set to 35, please use Android Studio Koala Feature Drop | 2024.1.2.

USB driver settings

Install the USB driver on the PC to connect to target Android device for application development. For USB driver settings, refer to the procedure of the Android device you are targeting to use as MOVERIO controller.

Connecting MOVERIO controler

This section describes how to connect the host devide to a computer with ADB (Android Debug Bridge) settings completed.

ADB is a versatile command line tool for communicating with devices, included in the Android SDK Platform-Tools package. See below for details.

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

Host Device setting

Please enable the developer settings of the host device. Then enable USB debugging. Check the detailed steps on the site below.

<https://developer.android.com/studio/debug/dev-options>

Connecting

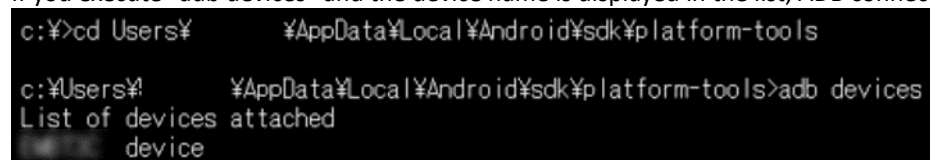
Use the ADB connection confirmation command to check if the computer and host device are connected.

Start a command prompt and run;

"cd C:\Users\<user name>\AppData\Local\Android\sdk\platform-tools"

*It is convenient to set the above path as an environment variable.

If you execute "adb devices" and the device name is displayed in the list, ADB connection is established



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

If it is not displayed, reconnect the host device with USB and execute "adb devices" again.

Embed the Moverio Basic Function SDK

Below explanation is based on using Android Studio.

Display the Project View of Android Studio and create a "libs" folder from [File]-[New]-[Directory].

After C:\Users\<user name>\AndroidStudioProjects\<application name>\app\libs is created.

Put BasicFunctionSDK_1.2.2.aar there.

Then define your application's minimum API level at 24 and add BasicFunctionSDK_1.2.2.aar to your application's dependencies.

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

build.gradle(Module: app)

```
android {
    :
    // define APIlevel 24
    minSdkVersion 24
    :
}

dependencies {
    :
    // add BasicFunctionSDK.aar
    implementation files('libs/BasicFunctionSDK_1.2.2.aar')
    :
    :
}
```

Push the Sync Project with Gradle Files button at the top of Android Studio to reflect Gradle changes to the project.

Note : If you want to set minifyEnabled property to true in build.gradle

If you build your application with Moverio Basic Function SDK included and minifyEnabled property in build.gradle set to true, Moverio Basic Function SDK may not work properly depending on the version of Android Studio. If you want to set the minifyEnabled property to true, please exclude the classes included in the Moverio Basic Function SDK package from obfuscation and optimization in your application's proguard-rules.pro file.

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

proguard-rules.pro(Module: app)

Display Control

Display brightness adjustment

Moverio is equipped with a see-through type display, and the visibility of the displayed image changes depending on the brightness of the surrounding environment. You can make the displayed image easier to see by increasing the brightness of the display when the surrounding environment is bright and by decreasing the brightness of the display when the surrounding environment is dark.

To adjust the display brightness, first establish communication with the Moverio display. To do this, create an instance of the `DisplayManager` class and call the `open()` method.

When using `DisplayManager` class of Moverio Basic Function SDK, in order to avoid name collision with `DisplayManager` class of Android standard, import “`com.epson.moverio.hardware.display.DisplayManager`” or use a class name of “`com.epson.moverio.hardware.display.DisplayManager`” at the time of instance generation.

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

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

Next, set the display brightness adjustment mode to manual mode or automatic mode. Use the `setBrightnessMode()` method to set it. Pass the `BRIGHTNESS_MODE_MANUAL` argument for manual mode and the `BRIGHTNESS_MODE_AUTOMATIC` argument for automatic mode. It cannot be used the `setBrightness()` method to adjust the display brightness when the display brightness adjustment mode is in automatic mode.

To adjust the display brightness manually, use the `setBrightness()` method to adjust the display brightness. For the adjustable display brightness range, refer to [Display Brightness Adjustable Range](#).

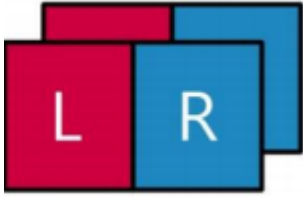
Finally, when you finish adjusting the brightness of your display, be sure to disconnect it from your Moverio display. Use the `close()` and `release()` methods to release.

```
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();
```

Switch 2D/3D display mode

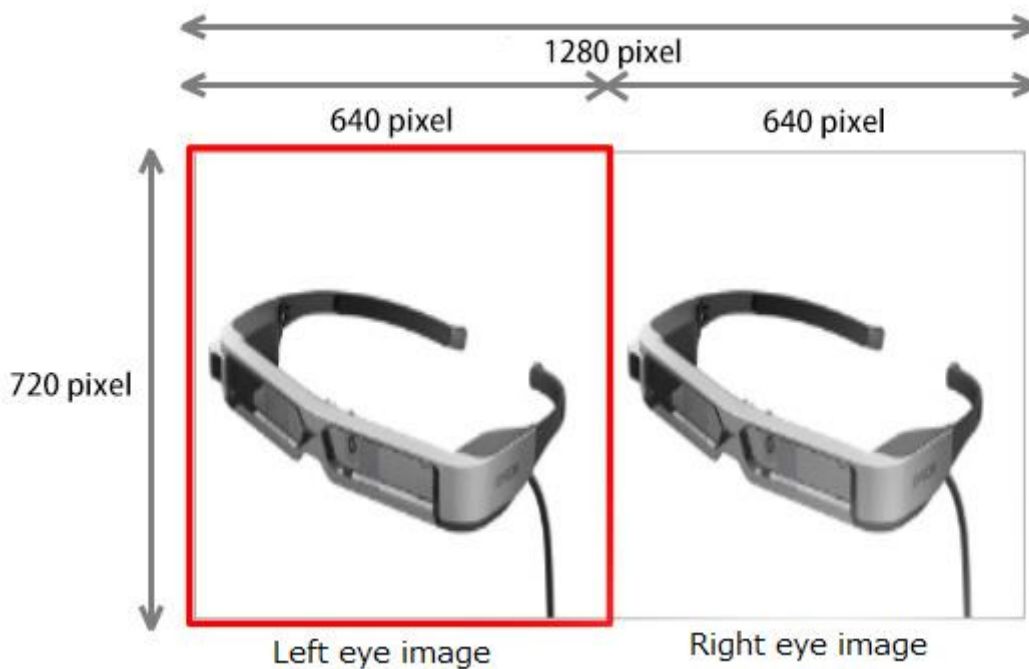
The Moverio display supports Side-by-Side 3D content display. If the video displayed by Moverio is Side-by-Side type 3D content and you want to control it from the app, use the dedicated API.

Side-by-Side format is a method to store image information for left eye and right eye side by side in one screen information.



When creating Side-by-Side format video, place the left-eye image and right-eye image in the left and right halves of the video, respectively.

For example, when creating a 1280x720 (HD size) Side-by-Side format image, place a 640x720 image in the left-eye image and a 640x720 image in the right-eye image as shown below.



To switch the display between 2D/3D display mode, first establish communication with the display of Moverio. To do this, create an instance of the DisplayManager class and call the open() method.

Next, set the 2D/3D display mode of the display to 2D display mode or 3D display mode. To set it, use the setDisplayMode() method. Pass the DISPLAY_MODE_2D argument for 2D display mode and the DISPLAY_MODE_3D argument for 3D display mode.

Finally, when you finish setting the 2D/3D display mode, be sure to disconnect the communication with the Moverio display. Use the close() and release() methods to release.

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

Display On/Off

If you want to hide the video displayed on Moverio, use the dedicated API.

To set the display on/off, first establish communication with the display on the Moverio. To do this, create an instance of the `DisplayManager` class and call the `open()` method.

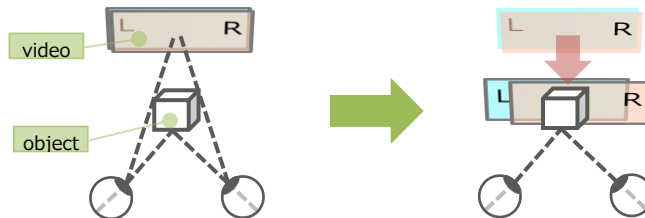
Then set the display to on or off. To set it, use the `setDisplayState()` method. Pass the `DISPLAY_STATE_ON` argument to show the display and the `DISPLAY_STATE_OFF` argument to hide the display. Finally, when you've completed On/Off your display, be sure to disconnect it from your Moverio display. Use the `close()` and `release()` methods to release.

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

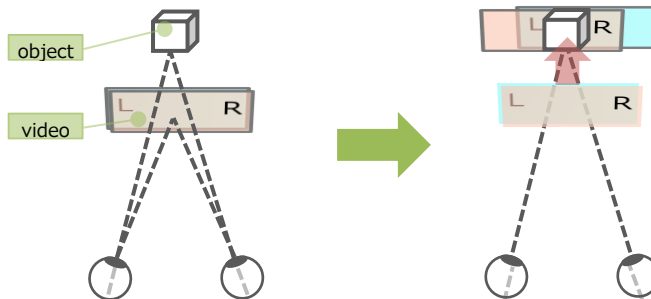
Display distance control

You can adjust the display distance of Moverio's display using a dedicated API. The dedicated API has a function to horizontally shift the left and right display images. You can adjust the display congestion according to the horizontal shift amount.

If the actual object is closer than the display distance, shift the left and right images inward as shown below. By doing so, the virtual display distance of the display can be reduced.



If the actual object is farther than the display distance of the display, shift the left and right images outward as shown below. By doing so, the virtual display distance of the display can be increased.



However, since the left and right images are shifted by using this function, the edges of the left and right images will be interrupted. Therefore, it is necessary for the application to consider the screen configuration according to the amount of video shift. For the relationship between the virtual display distance of the display and the image shift amount, refer to [The range of adjustable virtual display distance step and the horizontal pixel shift amount](#).

To adjust the display's virtual viewing distance, first establish communication with the Moverio display. To do this, create an instance of the DisplayManager class and call the open() method.

Next, set the adjustment amount step. For setting, use the setScreenHorizontalShiftStep() method. For the adjustment amount step, refer to [The range of adjustable virtual display distance step and the horizontal pixel shift amount](#).

Finally, when you're done making adjustments, be sure to disconnect your Moverio display. Use the close() and release() methods to release.

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

Display automatic sleep

You can set the automatic sleep setting of Moverio's display using a dedicated API.

When the display's automatic sleep setting is enabled, putting the Moverio on/off and placing it on a desk will turn off the display automatically. Also, the display will be displayed automatically when the MOVERIO headset is installed. By enabling this function, it is possible to reduce the power consumption when the Moverio headset is not installed.

To set the display to auto-sleep, first establish communication with the Moverio display. To do this, create an instance of the `DisplayManager` class and call the `open()` method.

Next, enable or disable the display's automatic sleep setting. Use the `setDisplayAutoSleepEnabled()` method for setting. Pass the `DISPLAY_AUTO_SLEEP_ENABLE` argument to enable the setting and the `DISPLAY_AUTO_SLEEP_DISABLE` argument to disable the setting.

Finally, when you're done with the settings, make sure you disconnect from the Moverio display. Use the `close()` and `release()` methods to release.

```
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();
```


Display manual sleep

A dedicated API can be used to set the user to manually put the Moverio display to sleep.

If you have enabled the display's manual sleep setting, tap the side of the Moverio to turn off the display. Also, tap the side of Moverio again to show the display. This function can be used when the wearer of Moverio wants to immediately turn off the front image.

To set the display to manual sleep, first establish communication with the display in Moverio. To do this, create an instance of the `DisplayManager` class and call the `open()` method.

Next, enable or disable the manual sleep setting on the display. Use the `setDisplayUserSleepEnabled()` method for setting. Pass the `DISPLAY_USER_SLEEP_ENABLE` argument to enable the setting and the `DISPLAY_USER_SLEEP_DISABLE` argument to disable the setting.

Finally, when you're done with the settings, make sure you disconnect from the Moverio display. Use the `close()` and `release()` methods to release.

```
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();
```

Creating images for the see-through function

MOVERIO is a device that uses projection technology. This system provides the user with an image projected onto a half-mirror via a light-guided panel, creating a half-mirror version (whereby not all the pixels are needed) allowing images to be arranged over a real-life scene giving a sense of transparency, and creating a more vivid augmented reality experience.

To create this transparent background effect, so visual elements (text, graphics...) stand out vividly, the background will need to be set to black when drawing on the projection, so you display the target section overlapping with the actual images.



Sensor Control

The MOVERIO has various sensors in the headset that detect movement, direction, and ambient illuminance. The Moverio Basic Function SDK allows you to get the raw data of these sensors. By using the sensor data, it is possible to guess the movement of the wearer's head and the surrounding brightness.

Sensor List

Moverio is equipped with various types of sensors.

There are hardware-based sensors such as Accelerometer, Geomagnetic Sensor, Gyroscope, and Ambient Light Sensor, as well as software sensors generated from the output values of multiple hardware-based sensors.

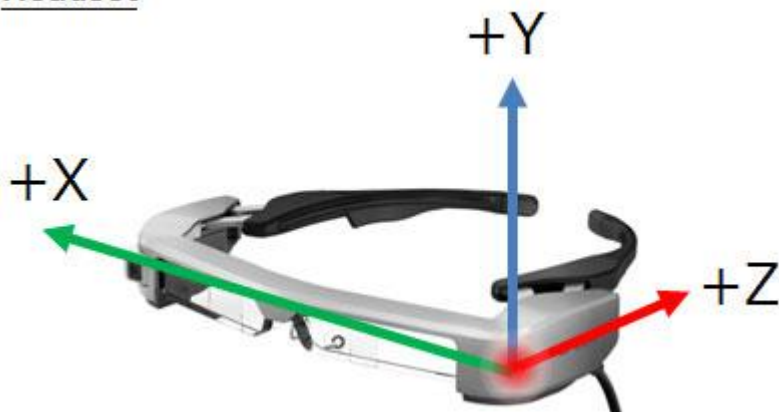
Sonser	Type	Discription	Usage
TYPE_ACCELEROMETER	Hardware	Measures the acceleration of Moverio including gravity with the acceleration [m/s ²] of 3 axes (x, y, z).	Motion detection (tilt, etc.)
TYPE_MAGNETIC_FIELD	Hardware	The surrounding geomagnetism is measured with 3-axis (x, y, z) geomagnetism [μT].	Orientation detection
TYPE_GYROSCOPE	Hardware	The angular velocity of Moverio is measured with the angular velocity [rad/s] of 3 axes (x, y, z).	Motion detection (rotation etc.)
TYPE_LIGHT	Hardware	Measure the ambient illuminance [lx].	Adjusting the brightness of the display according to the ambient illumination
TYPE_GRAVITY	Software	Gravity is measured by acceleration [m/s ²] on three axes (x, y, z).	Motion detection (tilt, etc.)
TYPE_LINEAR_ACCELERATION	Software	It is measured by acceleration of 3 axes (x, y, z) [m/s ²] excluding gravity.	Application to tap, walking detection, etc.
TYPE_ROTATION_VECTOR	Software	Measures the orientation of Moverio with a rotation vector.	Head tracking etc.
TYPE_MAGNETIC_FIELD_UNCALIBRATED	Software	The surrounding geomagnetism is measured with uncalibrated 3-axis (x, y, z) geomagnetism [μT] and 3-axis calibration information (x_bias, y_bias, z_bias).	Application to direction detection and head tracking
TYPE_GAME_ROTATION_VECTOR	Software	Measures the orientation of Moverio with a rotation vector that does not use geomagnetism.	Head tracking etc.
TYPE_GYROSCOPE_UNCALIBRATED	Software	The Moverio angular velocity is measured on the uncalibrated three-axis (x, y, z) angular velocity [rad/s] and the estimated drift three-axis (x_bias, y_bias, z_bias).	Application to motion detection (rotation etc.) and head tracking
TYPE_STATIONARY_DETECT	Software	Detects the Moverio stationary.	Headset attachment/detachment detection, etc.
TYPE_MOTION_DETECT	Software	It detects the movement of Moverio.	Headset attachment/detachment detection, etc.
TYPE_ACCELEROMETER_UNCALIBRATED	Software	The acceleration of Moverio including gravity is measured with the uncalibrated 3-axis (x, y, z) acceleration [m/s ²] and the estimated bias compensation 3-axis (x_bias, y_bias, z_bias).	Application to motion detection (tilt, etc.) and head tracking

TYPE_HEADSET_TAP_DETECT	Software	Detects taps on the headset.	Headset tap detection
-------------------------	----------	------------------------------	-----------------------

Axis of Sensors

Moverio's headset sensor is the same as the Android standard sensor coordinate system. When wearing the Moverio, the X axis points to the right, the Y axis points up, and the Z axis points to the wearer's direction.

Headset



Acquiring sensor values

With the Moverio Basic Function SDK, you can use various sensors installed in Moverio.

To get sensor data, first establish communication with the sensor in Moverio. To do this, create an instance of the `SensorManager` class and call the `open()` method. At that time, pass the sensor type and sensor data listener instance as arguments.

Please execute `SensorManager#open` after confirming that the image is displayed on Moverio after 10 seconds or more after connecting Moverio to the smartphone with USB. If you execute `SensorManager#open` within 10 seconds, communication may not be established normally.

When using the `SensorManager` class of the Moverio Basic Function SDK, in order to avoid name collision with the Android standard `SensorManager` class, import "`com.epson.moverio.hardware.sensor.SensorManager`" or set a class name as "`com.epson.moverio.hardware.sensor.SensorManager`" when creating an instance.

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

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

The type of sensor supported depends on the model. Check the supported sensor types by using the `getSupportedSensorList()` method.

```

@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());
}

```

The sensor data can be obtained at any time with the registered listener's `onSensorDataChanged()` method. The `onSensorDataChanged()` method is called very fast. In order to use the sensor efficiently in the application, do not execute the time-consuming processing in the `onSensorDataChanged()` method as much as possible. Finally, if you want to end the acquisition of sensor data, please be sure to cancel the communication with the sensor of Moverio. Use the `close()` and `release()` methods to release.

Sensor data can be obtained as an instance of the `SensorData` class. For details on the `SensorData` class, refer to the API reference.

```

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.
    }
}

```

Notes on sensor control

When controlling the sensor with Moverio Basic Function SDK, it can be used with a single application software. If you want to control the sensor with another application software, be sure to stop using the sensor control of the application software that is using the sensor control before using it.

If you use multiple sensors at the same time on your Android device, the sensor data acquisition frequency may decrease.

Camera control

Moverio has a camera in the headset that captures the front of the wearer. With Moverio Basic Function SDK, you can acquire video data from the camera and shoot still images/videos. By using the video data of the camera, it can be applied to the purposes such as marker recognition in addition to normal shooting.

Note: The camera function of Moverio Basic Function SDK may not work properly on Android 10 smartphones. Be sure to check the following operating conditions before using the camera function.

	targetSdkVersion 28 or later	targetSdkVersion 27 or older
Android 11	Work	Work
Android 10	Not work	Work
Android 9 or older	Work	Work

In addition, applications registered in Google Play after November 2021 must have targetSdkVersion set to 30 or higher.

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

In order to release an application that incorporates the camera function of Moverio Basic Function SDK on Google Play, the target Android version of the application must be 9 or lower, or the target Android version of the application must be 11.

Note: The CaptureStateCallback class will be deprecated. Please be sure to use the CaptureStateCallback2 class for Version 1.1.0 or later.

CAMERA Specification

	BT-35E	BT-45C
Effective pixel count	500 million pixel	800 million pixel
Capture format	RGB565/ARGB8888	RGB565/ARGB8888/YUY2/H.264
[Capture format]	[RGB565/ARGB8888]	[RGB565/ARGB8888]
Image resolution	640x480, 60fps/30fps/15fps	640x480, 60fps/30fps
/frame rate	1280x720, 60fps/30fps/15fps 1920x1080, 30fps/15fps 2592x1944, 15fps	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
Exposure compensation mode	auto/manual	auto/manual
Manual exposure compensation step	-5 ~ +5	-7 ~ +5
Auto focus mode	off	auto/manual
Manual focus	-	+30 ~ +3000 [mm]
Sharpness	0 ~ +128	-
Brightness	-127 ~ +127	0 ~ +255
Gain	0 ~ +255	+1024 ~ +16383
White ballance	auto /cloudy_daylight(6000K)/daylight(5500K) /fluorescent(4200K)/incandescent(3200K) /twilight(3500K)	auto /cloudy_daylight(6000K)/daylight(5500K) /fluorescent(4200K)/incandescent(3200K)) /twilight(3500K)
Power line frequency	50Hz/60Hz	50Hz/60Hz/Disable
Camera indicator mode	auto	auto/on/off

Preview function

Note: If the data format is H.264, it is not possible to preview images.

Note: If you want to display a preview using an instance of the Surface class, see [Display a preview using an instance of the Surface class as an argument](#).

You can use the Moverio Basic Function SDK to display a preview of the camera image installed in Moverio.

To display the preview, first establish communication with the camera of Moverio. To do this, create an instance of the CameraManager class and call the open() method. At that time, pass the SurfaceHolder of SurfaceView used for preview display as an argument.

Next, call the startCapture() method of the CameraDevice class to start the acquisition of camera image data. Then, by calling the startPreview() method of the CameraDevice class, the preview image is displayed in the SurfaceView of the SurfaceHolder passed by the argument of the open() method.

When using the CameraManager class of the Moverio Basic Function SDK, in order to avoid name collision with the AndroidManager CameraManager class, import com.epson.moverio.hardware.camera.CameraManager or class when creating an instance. Name it com.epson.moverio.hardware.camera.CameraManager

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

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

If you want to stop displaying the preview, call the stopPreview() method. When you finish using the camera, call the stopCapture() method to stop the acquisition of camera image data and call the close() and release() methods to disconnect the communication with the camera.

```
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);
        }

        :
        abridgement
        :

};

@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);
}
}

```

Acquisition of camera image data

Note: If you want to get the camera video data in ByteBuffer format, please refer to [Getting Camera Data in ByteBuffer Format](#).

With Moverio Basic Function SDK, you can get the video data of the camera installed in Moverio.

The acquisition of camera image data first establishes communication with the camera of Moverio. To do this, create an instance of the CameraManager class and call the open() method. At that time, pass an instance of CaptureDataCallback that receives the data as an argument. Then, by calling the startCapture() method of the CameraDevice class, you can receive the data with the instance of CaptureDataCallback passed in the argument of the open() method.

If you want to stop the acquisition of camera image data, call the stopCapture() method. When you finish using the camera, call the close() method and release() method to disconnect the communication with the camera.

If you move the app to the background while camera image data is being acquired, acquisition of camera image data will stop. To restart the acquisition of camera image data, call the startCapture() method again.

```
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);
        }

        :
        abridgement
        :
    };

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

        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);
}
}

```

Shooting still images/videos

Note: If the data format is H.264, it is not possible to shoot still/video images.

Note: Whenever you perform still/video recording, please refer to the following.

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

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

With the Moverio Basic Function SDK, you can shoot video from the camera installed in Moverio as a still image or video.

Shooting stills/videos first establishes communication with the camera of Moverio. To do this, create an instance of the `CameraManager` class and call the `open()` method. At that time, the instance of `CaptureStateCallback2` that receives the status notification of the completion of still image shooting and the start/end of video shooting is passed as an argument. Then call the `startCapture()` method of the `CameraDevice` class.

To take a still image, call the `takePicture()` method of the `CameraDevice` class. When the still image shooting is completed, you can receive the notification of the still image shooting completed by the `onPictureCompleted()` method of the instance of `CaptureStateCallback2` passed by the argument of the `open()` method.

Call the `startRecord()` method of the `CameraDevice` class to start video recording, and call the `stopRecord()` method of the `CameraDevice` class to end video recording. When the start/end of the video shooting, you can receive the notification of the start/end of the video shooting by the `onRecordStarted()` method and `onRecordStopped()` method of the instance of `CaptureStateCallback2` passed by the argument of the `open()` method. Video recording can be performed for up to 2 hours. If the video recording time exceeds 2 hours, the video recording will be forced to end. Also, if the storage capacity falls below 10% during movie recording, movie recording will be forcibly ended. Executing the `stopRecord()` method inside a method such as `onStop/onDestroy` in the Activity life cycle results in an error.

When you finish using the camera, call the `stopCapture()` method to stop the acquisition of camera image data, and then call the `close()` and `release()` methods to disconnect the communication with the camera.

Change camera properties

Note: The camera's exposure adjustment function should be used in Auto Exposure (AE). Use the manual exposure compensation mode only if the camera image cannot be acquired as expected with automatic exposure (AE).

Note: If the resolution is set to 640x480 and the exposure setting is brightened in a bright ambient light environment, camera image data may not be acquired.

With Moverio Basic Function SDK, you can change various properties of the camera installed in Moverio.

To change camera properties, first establish communication with the camera of Moverio. To do this, create an instance of the CameraManager class and call the open() method. At that time, pass an instance of SurfaceHolder or SurfaceView used for preview display or an instance of CaptureDataCallback that receives data. Next, call the getProperty() method of the CameraDevice class to get the current camera properties as an instance of the CameraProperty class. To change various camera properties, call the setXXX() methods of the instance of the CameraProperty class.

For example, if you want to change the exposure step of the camera, call the setExposureStep() method of the CameraProperty class. At that time, pass the exposure step of the changed camera as an argument.

Then, call the setProperty() method of the CameraDevice class to reflect the changed camera properties on the camera image. At that time, pass the instance of the CameraProperty class that is the changed camera property as an argument.

Camera properties can be changed even after capturing the camera image by calling the startCapture() method except for some.

Item	Before capturing	After captured
Capture format	able to change	unable to change
Image resolution/Frame rate	able to change	unable to change
Exposure compensation mode	able to change	able to change
Manual exposure compensation step	able to change	able to change
Auto focus mode	able to change	able to change
Manual focus adjustment	able to change	able to change
Sharpness	able to change	able to change
Brightness	able to change	able to change
Gain	able to change	able to change
White balance	able to change	able to change
Power line frequency	able to change	able to change
Camera indicator mode	able to change	able to change

The manual exposure compensation step can be changed according to the ambient illuminance [lx] of the usage environment. However, the exposure compensation step for high ambient illuminance may affect the frame rate of the camera image because the exposure time of the camera is lengthened to accommodate high ambient illuminance. Details are shown in the table below.

Manual exposure compensation step	BT-35E		BT-45C	
	ambient illuminance [lx]	Maximum frame rate [fps]	ambient illuminance [lx]	Maximum frame rate [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: The frame rate of the camera image of Moverio is 60[fps] at maximum. Also, the frame rate may decrease depending on the processing performance of the host device.

Capture format

Using the Moverio Basic Function SDK, you can change the data format output by the cameras installed in Moverio. To change the data format, see [Change camera properties](#).

RGB565/ARGB8888/YUY2 conforms to Android's ImageFormat specification.

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

H.264 conforms to the ITU-T specifications. <https://www.itu.int/rec/T-REC-H.264>

Display a preview using an instance of the Surface class as an argument

You can use the Moverio Basic Function SDK to display a preview of the camera image installed in Moverio.

To display the preview, first establish communication with the camera of Moverio. To do this, create an instance of the CameraManager class and call the open() method. At that time, pass the SurfaceView's SurfaceHolder to be used for preview display as null. Next, run the CameraDevice class' setPreviewSurface() method with the Surface instance as the argument. Next, call the startCapture() method of the CameraDevice class to start the acquisition of camera image data. Then, by calling the startPreview() method of the CameraDevice class, the preview image is displayed in the SurfaceView of the SurfaceHolder passed by the argument of the setPreviewSurface() method. If you want to stop displaying the preview, call the stopPreview() method. When you finish using the camera, call the stopCapture() method to stop the acquisition of camera image data and call the close() and release() methods to disconnect the communication with the camera.

```
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();
        }
        :
        abridgement
        :
    };

    @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);
}
}

```

Getting Camera Data in ByteBuffer Format

With Moverio Basic Function SDK, you can get the video data of the camera installed in Moverio.

The acquisition of camera image data first establishes communication with the camera of Moverio. To do this, create an instance of the CameraManager class and call the open() method. At that time, pass the CaptureDataCallback that will receive the data as null. Next, the CameraDevice class setCaptureDataCallback() method is executed with an instance of CaptureDataCallback2 as an argument. Then, by calling the startCapture() method of the CameraDevice class, you can receive the data with the instance of CaptureDataCallback2 passed in the argument of the setCaptureDataCallback() method.

If you want to stop the acquisition of camera image data, call the stopCapture() method. When you finish using the camera, call the close() method and release() method to disconnect the communication with the camera.

If you move the app to the background while camera image data is being acquired, acquisition of camera image data will stop. To restart the acquisition of camera image data, call the startCapture() method again.

```

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();
        }
        :
        abridgement
        :
    };

    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);
}
}

```

Permission request

When operating an application developed using the Moverio Basic Function SDK on an Android device for the first time, a permission request is sent to the user of the Android device. The required permissions are as follows.

Item	Timing of request
Camera (Android9 or later)	Calling at CameraManager#open() method
USB communication	Calling at CameraManager#open() method
Store still image/movie to storage (Android 10 or earlier)	Calling at CameraDevice#startCapture() method
Permission of audio recording in movie shooting	Calling at CameraDevice#startCapture() method

The application implements the ActivityCompat.OnRequestPermissionsResultCallback interface and receives that the user has granted the permission. At that time, call the onRequestPermissionsResult() method of the PermissionHelper class to notify the user's permission to the Moverio Basic Function SDK. When the Moverio Basic Function SDK is notified of the user's permission, the camera function can be used.

```
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);
    }
}
```

How to save still images/videos on SD card

To save the shooting results of still images/videos to the SD card, access the data storage area for applications in the SD card. Specify the following path in the takePicture() method of the CameraDevice class and the startRecord() method of the CameraDevice class.

[SD card path]/Android/data/[package name]/[file name]

Notes on camera control

Camera control using the Moverio Basic Function SDK can only be used with a single app. If you want to use the camera control with another app, be sure to stop using the camera control with the app that is using the camera control before using it.

Audio control

Moverio can output sound with earphones that conform to the CTIA standard. Moverio Basic Function SDK allows you to adjust the volume of the earphones.

Adjusting the earphone volume

You can increase or decrease the volume of the earphones according to the surrounding environment.

To adjust earphone volume, first establish communication with Moverio audio. To do this, create an instance of the AudioManager class and call the open() method.

When using the AudioManager class of the Moverio Basic Function SDK, in order to avoid name collision with the Android standard AudioManager class, import com.epson.moverio.hardware.audio.AudioManager or class when creating an instance. Name it com.epson.moverio.hardware.audio.AudioManager.

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

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

And to adjust the earphone volume, use the setVolume() method to adjust the earphone volume. For the earphone volume range that can be adjusted, refer to [Earphone Volume Range](#).

Finally, when you have finished adjusting the earphone volume, be sure to disconnect communication with Moverio's audio. Use the close() method to release it.

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

Adjusting the audio gain

Note: The use of the audio gain adjustment function may result in a high volume.

When using the function, always check the "Safety Instructions and Notices" section of the product manual.

The BT-45C's built-in speaker can amplify the output volume.

In order to use the audio gain adjustment feature, you need to declare a custom permission for audio gain adjustment in AndroidManifest.xml. The custom permission is the applicationId in build.gradle with ".permission.AUDIO_GAIN" concatenated to it. By using \${applicationId}, you can refer to the applicationId defined in build.gradle.

Note: To use the audio gain adjustment feature on Android 11 or later smartphones with BasicFunctionSDK Version 1.2.0, the permission-group must be defined as "com.epson.moverio.permission-group.MOVERIO" in AndroidManifest.xml.

Note: BasicFunctionSDK Version 1.2.1 or later does not require permission-group to be specified in AndroidManifest.xml when using the audio gain adjustment feature.

```
<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>
```

To adjust the audio gain, you first need to establish communication with Moverio's audio. To do this, create an instance of the AudioManager class and call the open() method.

When using the Moverio Basic Function SDK's AudioManager class, in order to avoid name collisions with the standard Android AudioManager class, please import com.epson.moverio.hardware.audio. AudioManager, or use com.epson.moverio.hardware.audio.AudioManager as the class name when creating the instance.

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

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

You can then adjust the audio gain by adjusting the volume of the earphones using the setGainStep() method. For the range of audio gains that can be adjusted, see [Audio gain range](#).

Finally, when you have finished adjusting the audio gain, be sure to disconnect the Moverio from the audio. Use the close() method to release it.

```

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);
    }
}

```

Setting the headset audio output mode

Note: If you wish to use the audio function of the BT-45C, you will need to grant access to the USB device twice.

Note: When you have completed the setting of the headset audio output mode, the communication with the Moverio audio will be automatically deactivated. If you wish to use the audio functionality provided by the MoverioBasicFunctionSDK again, be sure to call the `open()` method.

The BT-45C allows the user to set the headset audio output mode.

To set the headset audio output mode, first establish communication with Moverio audio. To do this, create an instance of the `AudioManager` class and call the `open()` method.

When using the `AudioManager` class of the Moverio Basic Function SDK, in order to avoid name collision with the Android standard `AudioManager` class, import `com.epson.moverio.hardware.audio.AudioManager` or class when creating an instance. Name it `com.epson.moverio.hardware.audio.AudioManager`.

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

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

The headset audio output mode can then be set using the `setDeviceMode()` method. The headset audio output modes that can be set can be found below.

	Description
<code>DEVICE_MODE_BUILTIN_AUDIO</code>	Built-in speaker mode
<code>DEVICE_MODE_AUDIO_JACK</code>	Audio jack mode

Finally, when you have completed setting the headset audio output mode, communication with Moverio's audio will be automatically deactivated. If you wish to use the audio features provided by the MoverioBasicFunctionSDK again, be sure to call the `open()` method.

```
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);
}
}
```


Device management overview

Moverio can detect important system state changes such as USB connection/disconnection with the host device, display start on the display, and device temperature abnormality. You can also use the device-specific information of Moverio. Moverio Basic Function SDK can detect changes in the system status and acquire device-specific information. By detecting changes in the system state, applications can detect that Moverio is connected and execute a function, or detect that the device has become hot and notify the user. Then, by using the device-specific information, it is possible to introduce a device authentication mechanism that allows only a specific Moverio to operate in the application.

Detect changes in system state

The Moverio Basic Function SDK can detect changes in the system status that are important for using Moverio, such as USB connection/disconnection, display start on the display, and device temperature abnormalities. By detecting changes in the system state, applications can detect that Moverio is connected and execute a function, or detect that the device has become hot and notify the user.

To detect system state changes, import `com.epson.moverio.system.HeadsetStateCallback`.

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

Implement the `HeadsetStateCallback` interface in the class that uses this function, and execute `registerHeadsetStateCallback`. `OnHeadsetAttached()` is called when the USB connection with the host device is detected, and `onHeadsetDetached()` is called when the USB removal is detected. Also, in order to start displaying the display and detect changes in the system status due to temperature abnormalities of various devices, it is necessary to create an instance of the `DeviceManager` class and execute the `open()` method. If the display of the Moverio display is started before the execution of the `DeviceManager#open()` method is completed, or if a temperature error occurs in various devices, the change in the system status cannot be detected. .. Finally, if you want to end the detection of changes in the system state by executing the `DeviceManager#open()` method, be sure to cancel the communication with Moverio. To release, use `close()` method, `release()` method and `unregisterHeadsetStateCallback()`.

```
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.
}
}

```

Acquisition of device-specific information

Moverio can use device-specific information such as model information and headset serial number. By using the device-specific information, you can introduce a device authentication mechanism that allows only a specific moverio to operate in the application.

To get device-specific information, first establish communication with Moverio. To do this, create an instance of the DeviceManager class and call the open() method.

Next, you can get device-specific information by using the dedicated API.

Use the getHeadsetProductName() method to get the model information. To get the headset serial number, first use the isHeadsetSerialNumberAcquisitionSupported() method to check that the connected Moverio can get the headset serial number, and then use the getHeadsetSerialNumber() method.

Finally, when you have finished obtaining device-specific information, be sure to cancel communication with Moverio. Use the close() and release() methods to release.

```

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();

```


Debug application of headset model and controller (Android smartphone, etc.) (network debugging)

When developing an application for a headset model (USB connection type Moverio headset) such as BT-40, confirm the application operation with a controller connected to USB such as Android smartphone. At this time, the PC and controller (Android smartphone, etc.) are not connected by USB, so wired debugging cannot be performed.

This section describes how to develop and debug application software for headset models and Android smartphones that are connected via USB by connecting to an Android smartphone via a network. Please check the following sites as well.

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

1. Connect your computer and Android smartphone to the same network.
2. Connect your computer and Android smartphone via USB.
*Hereafter, it will be described on the assumption that there is only one Android smartphone connected to the computer.
3. Execute the following command on the personal computer to enable adb connection to the Android smartphone via the network.

```
# adb tcpip 5555
```

4. Confirm the IP address of the Android smartphone.
On some Android smartphones, the IP address can be confirmed from the setting screen. The procedure up to the screen where the IP address is displayed varies depending on the Android smartphone. For details, please check the operating method of your Android smartphone.
5. Execute the following command on the personal computer to connect adb to the Android smartphone via the network.

```
# adb connect ip_address  
ex: adb connect 192.168.1.10
```

6. Terminate the USB connection between your computer and Android smartphone.
7. Execute the following command on the personal computer and confirm that the Android smartphone is connected to the personal computer by adb. Make sure the device you just connected is listed.

If not connected (not listed), check the following:

- Are your computer and Android smartphone connected to the same network?
- Repeat the procedure from step 3 several times
- Is the IP address entered in 6. correct?

```
# adb devices
```

8. Install the application on the Android smartphone and check the application operation using Android Studio.

Using the Moverio Basic Function SDK from Kotlin

This section shows how to use the Moverio Basic Function SDK from Kotlin.

Development with Moverio Basic Function SDK

Include the Moverio Basic Function SDK for projects where kotlin is selected as the development language.

Refer “Embed the Moverio Basic Function SDK” of “Android application software development procedure” in this document.

Example of running the Moverio Basic Function SDK API in Kotlin

Display control in Kotlin

The following is an execution example of brightness acquisition/setting.

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

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

displayManager.close()
```

Sensor control in Kotlin

This is an execution example of accelerometer data acquisition.

```
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.
    }
}
```

Camera control in Kotlin

The execution example of preview acquisition is described below.

```
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()
        }
        :
        abridgement
        :
    }

    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)
    }
}
```

About Android multi-display

Android supports screen output to multiple displays. There are two methods: "[Using the Presentaion class](#)" and "[Using the multi-display function of Android 8 or later](#)". "How to use the Presentaion class" can be used on devices that support API level 17 or later. "How to use the multi-display function of Android 8 or later" is available on compatible terminals of Android 8 or later. "How to use the multi-display function of Android 8 or later" can output the screen to Moverio without modifying the existing application. However, the input operation to the application depends on the terminal used. By utilizing these functions, output to the screen of Moverio and display on the screen of the terminal can be used.

Note: In "How to use the multi-display function of Android 8 or later", the activity of the existing application must support multi-window mode.

Summary of Moverio application software development for Windows

Below are the steps required to develop the BT-40/40S Windows applications for Moverio.

Install Windows SDK

The following describes how to install the Windows SDK on a computer equipped with Windows 10.

Acquire Visual Studio 2019

Download Visual Studio 2017 from the following site.

(As of Jul 2019, it is Visual Studio 2019) <https://visualstudio.microsoft.com/vs/>

Install Visual Studio 2019

Launch Visual Studio Installer and follow the instructions to install Visual Studio 2019.

Select workload

Start Visual Studio Installer and, select "Universal Windows Platform development", ".NET desktop development" and "Desktop development with C++" on the workload screen and install. Please check the detailed procedure on the site below.

<https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019>

When you are in trouble

Please check Microsoft's Visual Studio documentation.

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

Developing an application software

In order to create a desktop application. Choose the workload that fits your application.
Please check the detailed procedure on the site below.

<https://docs.microsoft.com/en-us/windows/win32/>

Display control on Windows

Moverio is equipped with a see-through (transmissive) display. The display control in the Windows desktop application can be controlled by sending a dedicated command using System.IO.Ports.SerialPort which is the standard COM port access API of Windows. You can control the brightness of the display and switching between 2D/3D display modes by using the dedicated commands. The brightness of the display can be adjusted from a dedicated command or can be automatically adjusted according to the illuminance of the surrounding environment. It is also possible to support the display of 3D contents in Side-by-Side format.

Display brightness adjustment on Windows

Moverio is equipped with a see-through type display, and the visibility of the displayed image changes depending on the brightness of the surrounding environment. You can make the displayed image easier to see by increasing the brightness of the display when the surrounding environment is bright and by decreasing the brightness of the display when the surrounding environment is dark.

To adjust the brightness of the display, use the standard Windows COM port access API System.IO.Ports.SerialPort and execute the dedicated command "setbright xx" (xx specifies 0 to 20). Please give me. The brightness of the display can be adjusted in 21 steps from 0 to 20. To set the display brightness adjustment mode to automatic mode, execute the dedicated command "setbright 50". If the dedicated command "setbright xx" (xx specifies 0 to 20) is executed when the display brightness adjustment mode is the automatic mode, the brightness adjustment mode switches to the manual mode.

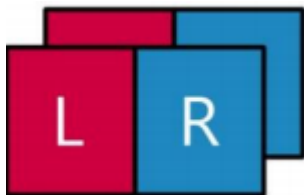
To get the current brightness of the display, execute the special command "getbright". The command returns 0 to 20 when the display brightness adjustment mode is manual mode, and the command returns 50 when the display brightness adjustment mode is automatic mode.

For the range of display brightnesses that can be adjusted, see [Display Brightness Adjustable Range](#).

Switch 2D/3D display mode on Windows

The Moverio display supports Side-by-Side 3D content display. If the video displayed by Moverio is Side-by-Side type 3D content and you want to control it from the app, use System.IO.Ports.SerialPort, which is the standard COM port access API for Windows, and execute the dedicated command "set2d3d 0" (2D display mode) or the dedicated command "set2d3d 1" (3D display mode). For how to use System.IO.Ports.SerialPort, refer to the document of SerialPort class of Microsoft Corporation.

Side-by-Side format is a method to store image information for left eye and right eye side by side in one screen information.



When creating Side-by-Side format video, place the left-eye image and right-eye image in the left and right halves of the video, respectively.

For example, when creating a 1280x720 (HD size) Side-by-Side format image, place a 640x720 image in the left-eye image and a 640x720 image in the right-eye image as shown below.

To switch between 2D/3D display on the display, use the Windows standard COM port access API `System.IO.Ports.SerialPort` and use the command `"set2d3d 0"` (2D display mode) or the command `"set2d3d 1"` (Please execute 3D display mode).

To get the current 2D/3D display mode status of the display, execute the command `"get2d3d"`. Returns 0 if the 2D/3D display mode of the display is 2D display mode, 1 if it is 3D display mode.

Display On/Off on Windows

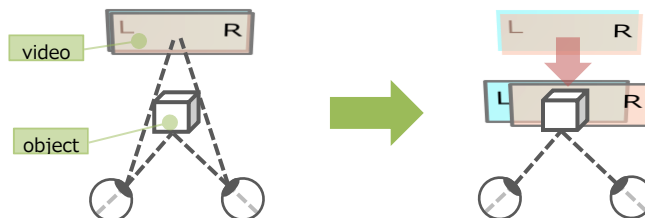
If you want to temporarily hide the video displayed in Moverio, use the standard Windows COM port access API `System.IO.Ports.SerialPort` and use the dedicated command `"setmute 0"` (display lit), Or execute the dedicated command `"setmute 1"` (display off). For how to use `System.IO.Ports.SerialPort`, refer to the document of `SerialPort` class of Microsoft Corporation.



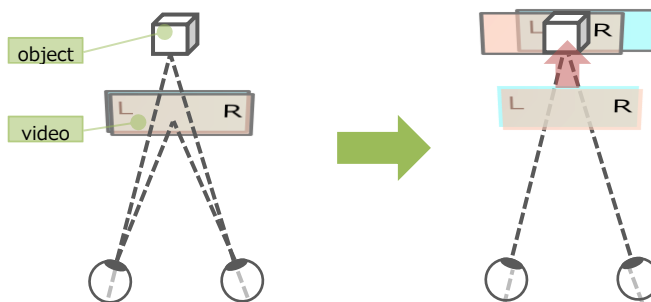
Display distance control on Windows

You can adjust the display distance of Moverio's display using a dedicated API. The dedicated API has a function to horizontally shift the left and right display images. You can adjust the display congestion according to the horizontal shift amount.

If the actual object is closer than the display distance, shift the left and right images inward as shown below. By doing so, the virtual display distance of the display can be reduced.



If the actual object is farther than the display distance of the display, shift the left and right images outward as shown below. By doing so, the virtual display distance of the display can be increased.



However, since the left and right images are shifted by using this function, the edges of the left and right images will be interrupted. Therefore, it is necessary for the application to consider the screen configuration according to the amount of video shift. For the relationship between the virtual display distance of the display and the image shift amount, refer to [The range of adjustable virtual display distance step and the horizontal pixel shift amount](#).

To adjust the virtual display distance of the display, use the standard Windows COM port access API System.IO.Ports.SerialPort, and use the dedicated command "setdisplaydistance xx" (xx is the pixel shift amount -32 to 256 Please specify). For how to use System.IO.Ports.SerialPort, refer to the document of SerialPort class of Microsoft Corporation.

For the pixel shift amount that can be specified with the dedicated command, refer to [The range of adjustable virtual display distance step and the horizontal pixel shift amount](#).

To get the current pixel shift amount, execute the dedicated command "getdisplaydistance".

Display automatic sleep on Windows

You can set the automatic sleep setting of the Moverio display using a dedicated command.

When the display's automatic sleep setting is enabled, putting the Moverio on/off and placing it on a desk will turn off the display automatically. Also, the display will be displayed automatically when the MOVERIO headset is installed. By enabling this function, it is possible to reduce the power consumption when the Moverio headset is not installed.

For the automatic sleep setting of the display, use the Windows standard COM port access API `System.IO.Ports.SerialPort` and use the dedicated command "enableautosleep 0" (autosleep OFF) or the dedicated command "enableautosleep 1". Execute (auto sleep ON).

To get the virtual display distance adjustment value of the current display, execute the special command "getautosleep".

Display manual sleep on Windows

A dedicated command can be used to set the user to manually put the Moverio display to sleep. ◦

If you have enabled the display's manual sleep setting, tap the side of the Moverio to turn off the display. Also, tap the side of Moverio again to show the display. This function can be used when the wearer of Moverio wants to immediately turn off the front image.

To set the display to manual, use the Windows standard COM port access API `System.IO.Ports.SerialPort`, and use the dedicated command "enableusersleep 0" (automatic sleep OFF) or the dedicated command "enableusersleep 1". Execute (auto sleep ON). For the usage of `System.IO.Ports.SerialPort`, refer to the document of Microsoft `SerialPort` class.◦

To get the virtual display distance adjustment value of the current display, execute the dedicated command "getusersleep".

Creating images for the see-through function

MOVERIO is a device that uses projection technology. This system provides the user with an image projected onto a half-mirror via a light-guided panel, creating a half-mirror version (whereby not all the pixels are needed) allowing images to be arranged over a real-life scene giving a sense of transparency, and creating a more vivid augmented reality experience. To create this transparent background effect, so visual elements (text, graphics...) stand out vividly, the background will need to be set to black when drawing on the projection, so you display the target section overlapping with the actual images.



Sensor control on Windows

The Moverio has various sensors in the headset that detect movement, direction, and ambient illuminance. Sensor control in Windows desktop apps uses the standard Sensor API. The sensor data can be used to estimate the wearer's head movement and sense the brightness of the surrounding environment.

Sensor list

Moverio is equipped with various types of sensors.

There are hardware-based sensors such as Accelerameter, Geomagnetic Sensor, Gyro Scope, and Ambient Light Sensor, as well as software sensors generated from the output values of multiple hardware-based sensors.

Sensor	Type	Description	Usage
SENSOR_TYPE_ACCELEROMETER_3D	Hardware	The acceleration of Moverio including gravity is measured by the acceleration [G] of 3 axes (x, y, z).	Motion detection (tilt, etc.)
SENSOR_TYPE_COMPASS_3D	Hardware	The surrounding geomagnetism is measured with 3-axis (x, y, z) geomagnetism [mG].	Orientation detection
SENSOR_TYPE_GYROMETER_3D	Hardware	The angular velocity of Moverio is measured with the angular velocity [deg/s] of 3 axes (x, y, z).	Motion detection (rotation etc.)
SENSOR_TYPE_AMBIENT_LIGHT	Hardware	Measure the ambient illuminance [lx].	Adjusting the brightness of the display according to the ambient illumination
GUID_SensorType_GravityVector	Software	Gravity is measured with acceleration [G] on three axes (x, y, z).	Motion detection (tilt, etc.)
GUID_SensorType_LinearAccelerometer	Software	It is measured by acceleration [G] on three axes (x, y, z) excluding gravity.	Application to tap, walking detection, etc.
GUID_SensorType_RelativeOrientation	Software	Measure the orientation of Moverio with a quaternion that does not use geomagnetism.	Head tracking etc.
SENSOR_TYPE_AGGREGATED_DEVICE_ORIENTATION	Software	Measure the orientation of Moverio with a quaternion.	Head tracking etc.
SENSOR_TYPE_CUSTOM(0)*1	Hardware	Measures the acceleration of Moverio including gravity with uncalibrated 3-axis (x, y, z) acceleration [G] and 3-axis of estimated bias compensation (x_bias, y_bias, z_bias).	Application to motion detection (tilt, etc.) and head tracking
SENSOR_TYPE_CUSTOM(1)*1	Hardware	The Moverio's angular velocity is measured on the uncalibrated 3-axis (x, y, z) angular velocity [deg/s] and the estimated drift 3-axis (x_bias, y_bias, z_bias).	Application to motion detection (rotation etc.) and head tracking
SENSOR_TYPE_CUSTOM(2)*1	Hardware	The surrounding geomagnetism is measured with uncalibrated 3-axis (x, y, z) geomagnetism [mG] and 3-axis calibration information (x_bias, y_bias, z_bias).	Application to direction detection and head tracking
SENSOR_TYPE_CUSTOM(3)*1	Software	Detects the movement/movement of Moverio.	Headset attachment/detachment detection, etc.
SENSOR_TYPE_CUSTOM(4)*1	Software	Detects taps on the headset.	Headset tap detection

*1 In the case of [SENSOR_TYPE_CUSTOM](#), it is determined by comparing it with the value of [SENSOR_DATA_TYPE_CUSTOM_USAGE](#) and the value in parentheses in the above table.

Check the site below for more information.

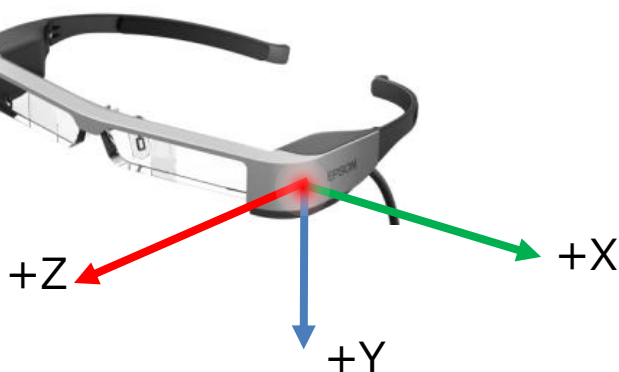
<https://docs.microsoft.com/en-us/windows/desktop/sensorsapi/sensor-categories--types--and-datafields>

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

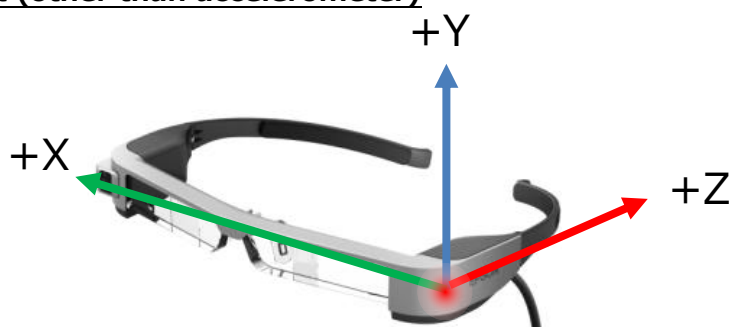
Axis of Sensors

The coordinate system of Moverio's sensor differs between the accelerometer and other sensors. When wearing the Moverio, the X-axis of the accelerometer points to the left, the Y-axis points down, and the Z-axis points to the wearer's line of sight. Except for the accelerometer, the X axis points to the right, the Y axis points up, and the Z axis points to the wearer.

Headset (for accelerometer)



Headset (other than accelerometer)



Camera control on Windows

Moverio has a camera in the headset that captures the front of the wearer. Camera control in Windows desktop apps uses Microsoft Media Foundation, ([Link](#)) which is a standard Windows camera control API. With this API, you can shoot movies and display previews.

CAMERA Specification

	BT-35E	BT-45C
Effective pixel count	500 milion pixel	800 milion pixel
Capture format	RGB565/ARGB8888	RGB565/ARGB8888/YUY2/H.264
[Capture format]	[RGB565/ARGB8888]	[RGB565/ARGB8888]
Image resolution	640x480, 60fps/30fps/15fps	640x480, 60fps/30fps
/frame rate	1280x720, 60fps/30fps/15fps	1280x720, 60fps/30fps
	1920x1080, 30fps/15fps	1920x1080, 30fps
	2592x1944, 15fps	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
Exposure compensation mode	auto/manual	auto/manual
Manual exposure compensation step	-5 ~ +5	-7 ~ +5
Auto focus mode	off	auto/manual
Manual focus	-	+30 ~ +3000 [mm]
Sharpness	0 ~ +128	-
Brightness	-127 ~ +127	0 ~ +255
Gain	0 ~ +255	+1024 ~ +16383
White ballance	auto	auto
	/cloudy_daylight(6000K)/daylight(5500K)	/cloudy_daylight(6000K)/daylight(5500K)
	/fluorescent(4200K)/incandescent(3200K)	/fluorescent(4200K)/incandescent(3200K)
	/twilight(3500K)	/twilight(3500K)
Power line frequency	50Hz/60Hz	50Hz/60Hz/Disable
Camera indicator mode	auto	auto/on/off

Audio control on Windows

Moverio can connect a CTIA-compliant earphone/microphone for use in listening to sound sources and voice calls. To control audio in Windows desktop apps, use Core Audio APIs ([Link](#)), which is the standard audio control API of Windows. With this API, you can perform earphone output of sound source, microphone input of voice, various parameter settings of earphone/microphone, etc.

Earphone output of sound source

With Moverio, you can listen to the sound source from the connected earphones using Core Audio APIs. Check the site below for more information.

<https://docs.microsoft.com/en-us/windows/desktop/coreaudio/rendersharedeventdriven>

Microphone input for voice

With Moverio, you can input audio from a microphone connected using Core Audio APIs. Check the site below for more information.

<https://docs.microsoft.com/en-us/windows/desktop/coreaudio/capturesharedeventdriven>

Parameter settings

Moverio allows you to set various parameters for earphones and microphones that are connected using Core Audio APIs. Check the site below for more information.

<https://docs.microsoft.com/en-us/windows/desktop/CoreAudio/programming-guide>

Mute setting

With Moverio, it is possible to temporarily stop (mute) earphone output and microphone input using Core Audio APIs. Check the site below for more information.

<https://docs.microsoft.com/en-us/windows/desktop/api/audiopolicy/nf-audiopolicy-iaudiosessionevents-onsimplevolumechanged>

Device management overview on Windows

Moverio can detect important system state changes such as USB connection/disconnection with the host device, display start on the display, and device temperature abnormality. In addition, Moverio has device-specific information that can be used. By monitoring changes in the system state, applications can detect when the device is hot and notify the user. By using the device-specific information, you can build and implement a device authentication mechanism that allows only a specific moverio to operate in the application.

Moverio model discrimination

Moverio has different functions depending on the model, and the mounted camera, sensor type, etc. are different. Therefore, the model of the Moverio connected to the Windows terminal can be identified using the Vendor ID and Product ID of the USB CDC (Communication Device Class) information. See the table below for details.

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

To obtain USB CDC information of Moverio connected to Windows device, use ManagementClass class of .NET API. Specify "Win32_SerialPort" when creating an instance of this class, get the information of "PNPDeviceID" for each port, and determine the model by comparing with the above table.

```
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");
    }
}
```

Check the detailed information of ManagementClass class on the following site.

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

Monitor system state changes

With Windows, you can monitor changes in important system status of Moverio, such as the start of display, temperature abnormalities of various devices, etc. By monitoring changes in the system state, applications can detect when the device is hot and notify the user.

To monitor changes in the system status, use the Windows standard COM port access API `System.IO.Ports.SerialPort` and execute the command "getsystemstat" (get system status). For how to use `System.IO.Ports.SerialPort`, refer to the document of `SerialPort` class ([Link](#)) of Microsoft Corporation.

The command return value is shown in the table below.

Return value	Description
0	Power off
1	In a process of initialing the Moverio system
2	There is no video output from the host device
3	During image display
4	Display off
5	In recovery process *In the automatic recovery function when the display goes out due to static electricity

Acquisition of device-specific information

Moverio can use the headset serial number information. By using device-specific information, it is possible to introduce a device authentication mechanism that allows only a specific moverio to operate in an application.

To obtain device-specific information, use `System.IO.Ports.SerialPort`, which is a Windows standard COM port access API, and execute the command "getserial" (get headset serial number). For how to use `System.IO.Ports.SerialPort`, refer to the document of `SerialPort` class ([Link](#)) of Microsoft Corporation.

Appendix

Display Brightness Adjustable Range

The display brightness adjustment range that can be selected differs depending on the model.

Model	Range of brightness
BT-35E/30E	0 ~ 20
BT-30C	0 ~ 20
BT-40	0 ~ 20
BT-45C	0 ~ 20

The range of adjustable virtual display distance step and the horizontal pixel shift amount

Refer to the table below for the virtual display distance range and horizontal shift amount that can be adjusted.

The display distance is a reference value, not a guaranteed value.

※BT-35E/30E and BT-30C are not supported.

Step	BT-40 Pixel shift amount	Display distance [m] reference value	BT-45C Pixel shift amount	Display distance [m] reference value
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

Earphone volume range

Adjustable earphone volume range

	range	
BT-35E/30E	0 ~ 15	
BT-30C	0 ~ 20	
BT-40	Not support	
BT-45C	Not support	

Audio gain range

Adjustable audio gain range

	range	
BT-35E/30E	Not support	
BT-30C	Not support	
BT-40	Not support	
BT-45C	0 ~ +4 (default=0)	