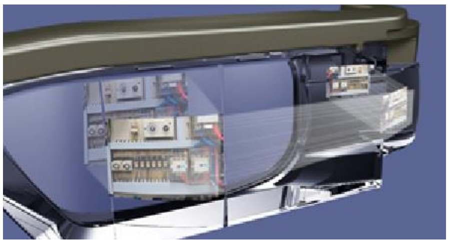
3. Display control

3.1. Display control summary

This chapter explains the display control function for the BT-2000.



The BT-2000 is a smart headset with an optical see-through function.

You can overlay information on the display using an optical technique that provides clear images, without disturbing the view of the user's surroundings. It is also possible to project more information since the device uses a see-through system allowing images to be projected for both eyes, as opposed to the single image one-eye type. The advantage with this device is that eye movement is kept to a minimum, thereby reducing fatigue and making it suitable for use at work.

You can also flip-up the MOVERIO Pro display, which allows you to set the display section by holding it with one hand when you want to superimpose video or images on your current work space.

The following section explains the functions and usage methods available.

- Full screen display function
- Switch between 2D/3D display function (API)
- Mute function (API)
- Backlight control
- See-through function

3.2. Full screen display function

The MOVERIO Pro BT-2000 is based on Android 4.0 Tablet UI. This specification does not normally allow full display in applications in Android 4.0 Tablet UI; however, you can follow the steps below to enable full display by specifying a unique flag in the app.

■ Executing full screen in applications

Execute the following process in onCreate() for each Activity.

```
Window win = getWindow();
WindowManager.LayoutParams winParams = win.getAttributes();
winParams.flags 0x80000000
win.setAttributes(winParams);
```

For apps with multiple Activities, execute the above process for each Activity.

■ Add an import definition

```
import android.view.Window;
import android.view.WindowManager;
```

Disable full screen display (show status bar)

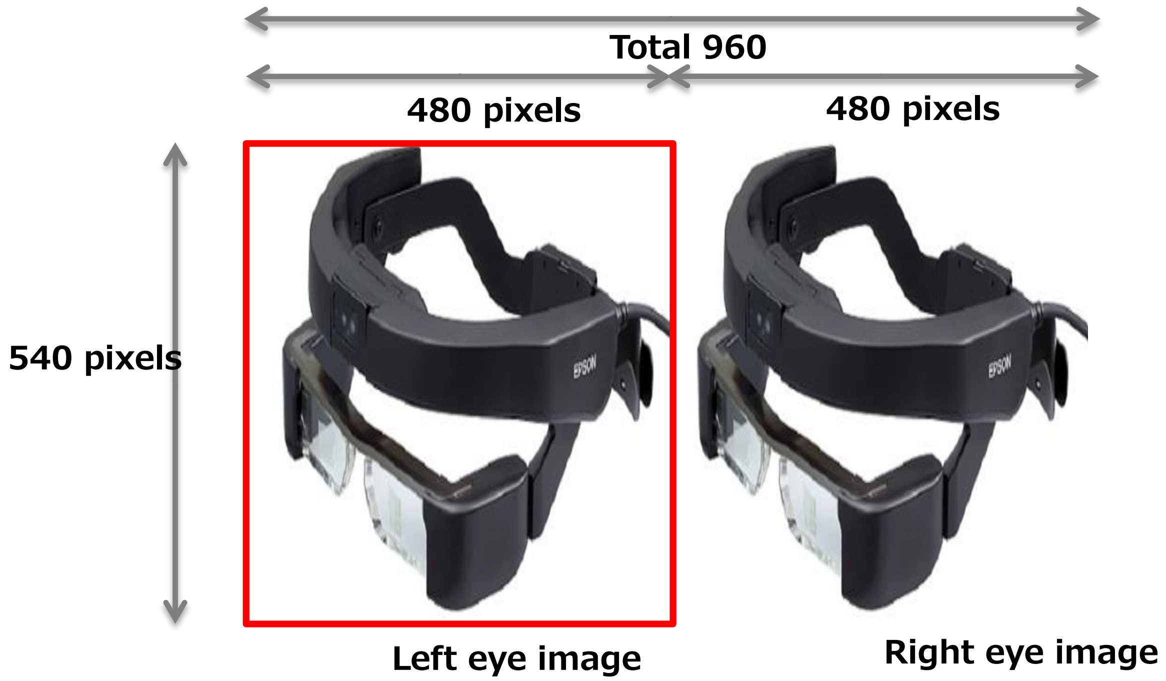
Enable full screen display (hide status bar)

3.3. Switch between 2D/3D display function

The BT-2000 allows you to display 3D content using Side by side.

The side by side method places images on the left and right of the screen.

When realizing the side by side system with QHD size screen, you need to arrange images from left to right by reducing 960 x 540 by half(480 x 540 pixels) to create one frame of a QHD image.



You can use the following interface to separate images for the left and right eyes in the side by side system, and output each display.

■**Inport file**

jp.epson.moverio.H725.DisplayControl

■**Constructor**

DisplayControl(Context context)

■ **Interface**

int setMode(int DisplayMode, Boolean toast)

■ **Parameter**

DisplayMode : 2D/3D mode situation

2D Mode : DisplayControl.DISPLAY_MODE_2D

3D Mode : DisplayControl. DISPLAY_MODE_3D

toast : Switch display/un-display of OSD of 2D/3D

Display : true

Un-display : false

3.4. Backlight control

3.4.1. Adjust backlight brightness

When changing the backlight display built-into the headset, you can change the transparency of the displayed image.

When the brightness is low, the image is more transparent, and when the brightness is high, the image is more opaque.

■ Import file

jp.epson.moverio.H725.DisplayControl

■ Constructor

DisplayControl(Context context)

■ **Interface**

Display brightness settings

int setBacklight(int backlight)

Display brightness acquisition

int getBacklight()

■ **Parameter**

backlight: Brightness level 0 (dark) to 20 (bright)

■ Return value

Run result 0 (Success), -1(Failure)

3.5. Mute function

3.5.1. Display mute

The mute function allows you to temporarily stop displaying images.

Use the following interface to activate and then deactivate mute.

■ **Import file**

jp.epson.moverio.H725.DisplayControl

■ **Constructor**

DisplayControl(Context context)

■ **Interface**

int setMute(boolean mute)

■ **Parameter**

mute: Mute ON (TRUE)/OFF (FALSE)

■ **Return value**

Execution result 0 (normal value), any other value (error)

3.5.2. Audio mute

The audio mute function allows you to temporarily stop outputting audio.

Use the following interface to activate and then deactivate mute.

- **Import file**

  jp.epson.moverio.H725. AudioControl

- **Interface**

  int setMute(boolean mute)

- **Parameter**

  mute: Mute ON (TRUE)/OFF (FALSE)

- **Return value**

  Execution result 0 (normal value), any other value (error)

- **Attentions**

When using int setMute(boolean mute), designate below permission to the Manifest file of App.

<uses-permission

android:name="android.permission.MODIFY_AUDIO_SETTINGS"></uses-permission>

3.6. See-through function

MOVERIO Pro is a device that uses projection technology.

Projection is displayed by reflecting light through an Active Matrix LCD panel into a light-guided panel. It is possible to create 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.

(In this situation, we recommend that you do not use the shade.)

The following steps allow you to create this AR effect using the see-through function.

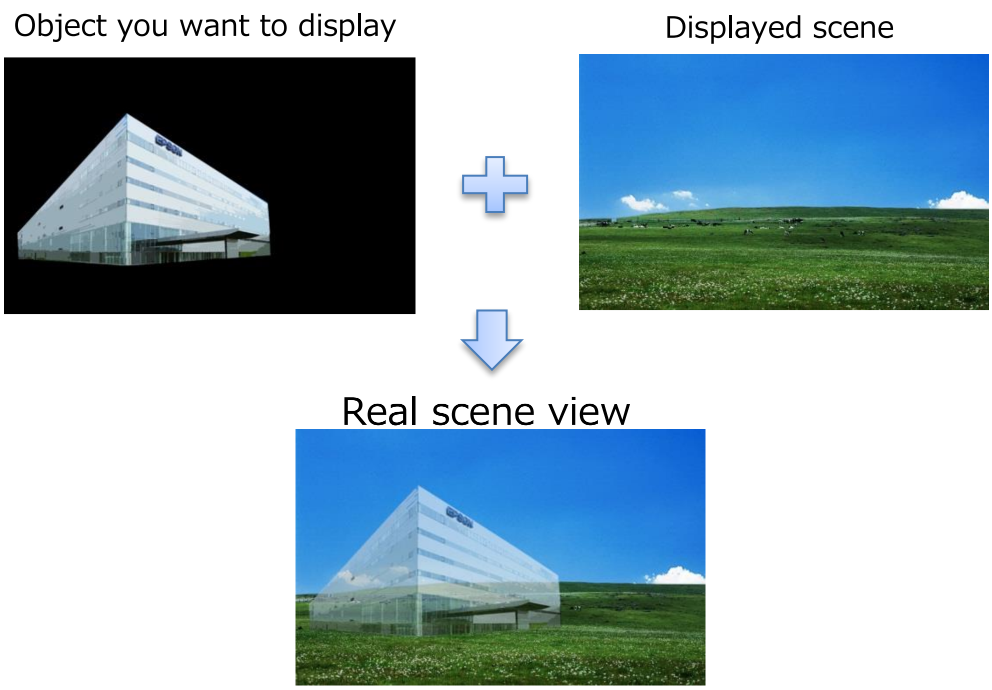
1) Execute full screen support.

   To reduce the feeling of being in a screen, remove everything except for the necessary image (status bar etc…).

2) Make the background black.

   Make everything black except for the object you want to display.

   In theory, the black section should keep out external light.



4. UI control

4.1. UI control summary

The user interface for the BT-2000 is composed of 11 hardware buttons; power, key lock, A, B, X, Y, D-pad (up, down, left, right), and the select/OK key. This section explains the functions called when these buttons are pressed, as well as the key assignment change function that is unique to the BT-2000.

4.1.1. Hardware button types and functions

The BT-2000 comes with the hardware buttons shown in figure 4-1. The functions for each button are shown in table 4-1.

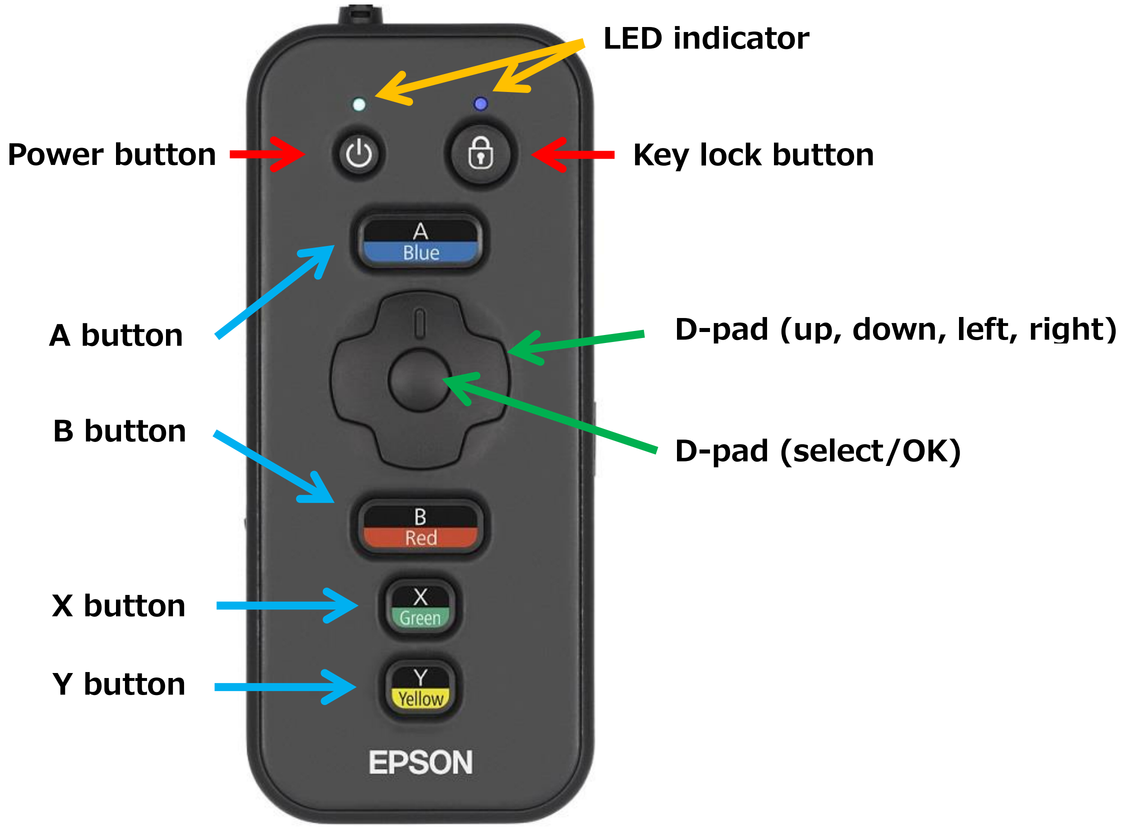


Figure 4-1 BT-2000 built-in hardware button

Table 4-1 Button name and its function

Button name	*Function*
Power	Turn the device ON/OFF
Key lock	Enable/disable button input
A button	Back
B button	Home
X button	Menu
Y button	MultiFunctionOSD display (See table 4-2)
D-pad up	Upper input
D-pad down	Lower input
D-pad left	Left input
D-pad right	Right input
Select/OK key	Confirm

Table 4-2: MultiFunctionOSD display function

MultiFunction operations

Number of times button pressed	Function	Adjustment method
Press once	adjust volume	D-pad up/right: Volume Up D-pad down/left: Volume Down
Press twice	Adjust brightness	D-pad up/right: Increase screen brightness D-pad down/left: Decrease screen brightness
Press three times	Switch between 2D/3D	D-pad up/right: Set 2D D-pad down/left: Set 3D

4.1.2. Software keyboard

BT-2000 has iWnnIME as standard character input function. When inputting character, below keyboard will be displayed and can control by D-pad key.

English keyboard

■Limited items

「SYM」key cannot be used by D-pad key. Please connect and use mouse for using「SYM」key.

4.1.3. BT-2000 change key assignment function

The BT-2000 allows you to change the functions assigned to the hardware buttons mentioned above. The standard hardware button assignments are applied in Default mode, the changed hardware button assignments are applied in User mode.

Default mode

Button name	Function
Power	Turn the device ON/OFF
Key lock	Enable/disable button input
A button	Back
B button	Home
X button	Menu
Y button	Multi Function OSD Display
D-pad up	Move up
D-pad down	Move down
D-pad left	Move left
D-pad right	Move right
select/OK key	Confirm

Change mode

User mode

Button name	Function
Power	Turn the device ON/OFF
Key lock	Enable/disable button input
A button	F1
B button	F2
X button	F3
Y button	F4
D-pad up	Move up
D-pad down	Move down
D-pad left	Move left
D-pad right	Move right
select/OK key	Confirm

■Events called when buttons are pressed

The following key events are generated.

Hardware button	Default mode Key event	User mode Key event
A button	KeyEvent.KEYCODE_BACK	KeyEvent.KEYCODE_F1
B button	KeyEvent.KEYCODE_HOME	KeyEvent.KEYCODE_F2
X button	KeyEvent.KEYCODE_MENU	KeyEvent.KEYCODE_F3
Y button	(Reserved in the system)	KeyEvent.KEYCODE_F4
D-pad up	KeyEvent.KEYCODE_DPAD_UP	
D-pad down	KeyEvent.KEYCODE_DPAD_DOWN	
D-pad left	KeyEvent.KEYCODE_DPAD_LEFT	
D-pad right	KeyEvent.KEYCODE_DPAD_RIGHT	
select/OK key	KeyEvent.KEYCODE_DPAD_CENTER	

■Using the change function for key assignments

By changing the key assignments, the key codes called when you press the A, B, X, or Y buttons are not the standard Back, Home, or Menu buttons in Android, instead, the key codes are changed to F1 to F4.
Therefore, you can include optional functions to the F1 to F4 codes making use of the A, B, X, and Y buttons as with the application's original function.

■ Specifications for status transitions in key assignment mode

Status transitions (where the value of the key is changed) for the key assignment mode cannot be set in the app; they should be set in the system. Management of key assignment mode must be done by last application before switched. For example, when the key assignment mode for application A is set to User mode, and then a different application B is started, the key assignment mode for application B is also set to User mode.

■Specifications for status transitions in key assignment mode when using iWnnIME

The BT-2000 is equipped with a standard iWnnIME text input system. This system uses the Back, Menu, and D-pad keys for input, and automatically changes the key assignments to Default mode when starting up the BT-2000. When closing an app, the key assignment mode changes to the mode set in iWnnIME for the last app that was started, however operations are not guaranteed when an error occurs or if the app does not close normally.
When an app is constructed with multiple source calls, and calls that apply not just iWnnIME, we recommend managing the key assignment status according to each app that is started.

■Notes on switching key assignment during key operation
Switching key assignment during pressing and holding the key may cause the wrong key events to be recognized as it pressed. Don’t call the switching API when the key is pressed for a long time.

■Notes on function assignment to the simultaneous key press
If you hold down "up"+"down"+"OK" or "left"+"right"+"OK" at the same time, the key lock will operate. Please do not assign function to press this button simultaneously.

4.2. Application interface list

Tables 4-3 provide a list of application interfaces for changing key assignments.

You also need to import the following models to use each API.

android.btutil.KeyAssign

Table 4-3 android.btutil.KeyAssign for application interface list

NO.	Function name	Function summary	Notes
1	getKeyAssignMode	Acquires the mode for the current key assignment.	
2	setKeyAssignMode	Set and apply a key assignment mode.	

4.3. Application interface details

4.3.1. getKeyAssignMode

- Function

Acquire the mode for the current key assignment.

- Format

```
static int getKeyAssignMode (void);
```

- Parameter

None.

- Return value

Model	Explanation
Int	- KEYASSIGN_MODE_DEFAULT: Default mode - KEYASSIGN_MODE_USER: User mode

- Usage procedure

```
int mode;
/*Acquire the current mode*/
mode = KeyAssign.getKeyAssignMode ();

if (mode == KeyAssign.KEYASSIGN_MODE_USER) {
     /*Process*/
}
```

4.3.2. setKeyAssignMode

- Function

  Set key assignment mode and apply to the system.

- Format

```
boolean setKeyAssignMode(int mode);
```

- Parameter

● Model name	Explanation
int mode	Key assignment mode to be set. You can specify the value using a macro for the following android.btutil.KeyAssign classes. - KEYASSIGN_MODE_DEFAULT (or 0) - KEYASSIGN_MODE_USER (or 1)

- Return value

  true: setting succeeded

  false: setting failed

Use procedure 1

```
/*Set user mode*/
KeyAssign.setKeyAssignMode (KeyAssign.KEYASSIGN_MODE_USER);
```

5. Voice commands

5.1. Pre-cautions for developing Voice commands Apps

Please make sure to confirm the version of SDK provided by EPSON and the version of system software inside BT-2000.

If the version of SDK provided by EPSON used for Apps development and the version of system software is not matching, due to the difference of API included, developed Apps may not operate.

※User using system software version R1.0.4

When updating OS from R1.0.4 to R1.2.1 or later, adding callback API process which is newly added is required.

This phenomenon may occur in below Apps.

・Apps using Voice command class(VoiceCommandClientCallbacks)

If it happens, please solve by below procedure.

1） Change the SDK of Apps development environment to SDK that matching system software version.

2） Revise the source code according to SDK and re-build.

5.2. Voice commands summary

The BT-2000 voice command will select the voice information file(lms file), and recognizes the voice included in this information, and returns the corresponding ID. Voice is inputted using a microphone, and the device recognizes the voice that has been registered*.

By mounting a process in the application for each ID acquired by callback, voice input can be used as a trigger.

* Only vocabulary that is included in the voice information file can be recognized, this differs from the standard sound recognition software that uses network access to confirm words.

5.3. Application interface function summary

You can use the following functions by using audio command API.

（1） Connect to a service class that provides audio command functions.

（2） Disconnect from a service class that provides audio command functions.

（3） Acquire an interface to control audio commands.

（4） Register a callback.

（5） Cancel a callback.

（6） Set parameters used for the audio recognition start conditions.

（7） Enable audio input status.

（8） Disable audio input status.

（9） Receive the recognition result of voice commands

（１０） Receive the situation of if voice commands can receive the input

To use API, execute VoiceCommandClient.bindToRemoteRunningService() first and connect to a service class that provides a voice command function. Next, execute VoiceCommandClient.registerCallback() to register a callback to receive recognition results. When connection to the service is complete, execute getVoiceCommandSystemInterface() to acquire the interface for controlling the voice command function.

5.4. Application interface list

Tables 5-1 and 5-2 provide a list of application interfaces for using the voice command function.

You also need to import the following models to use each API.

android.media.epson.IVoiceCommandInterface

android.media.epson.IVoiceCommandServiceCallbacks

android.media.epson.VoiceCommandClient

android.media.epsonVoiceCommandClientCallbacks

Table 5-1 android.media.epson.IVoiceCommandInterface for application interface list

NO.	Function name	Function summary
1	setSnr	Set parameters used for the voice recognition start conditions. Set the recognition start conditions according to the amount (S/N ratio) of surrounding voice (noise) compared with the input voice.
2	getSnr	Acquire the current value for the set S/N ratio.
3	setAmp	Set parameters used for the voice recognition start conditions. Set the recognition start conditions according to the amplitude for the input voice.
4	getAmp	Acquire the setting for the amplitude of the current input voice.
5	start	Enable voice input.
6	stop	Disable voice input.

Table 5-2 android.media.epson. VoiceCommandClient for application interface list

NO.	Function name	Function summary
1	registerCallback	Register a callback to acquire the recognition results for a voice command.
2	unRegisterCallback	Cancel a callback to acquire the recognition results for a voice command.
3	getVoiceCommandSystemInterface	Acquire an IVoiceCommandInterface class as the interface to control voice commands.
4	bindToRemoteRunningService	Connect to a service class of voice commands.
5	unBindFromRemoteRunningService	Disconnect from a service class of voice commands.

Table5-3 android.media.epson. VoiceCommandClientCallbacks application interface list

NO.	Function name	Function summary
1	onServiceConnected	Called after connection to service class implementing callback of voice command function has completed.
2	onServiceDisconnected	Called when service controlling voice command is stopped for some reasons.
3	onVoiceCommand	Callback for receiving voice command recognition result.
4	onSpeakable	Callback for receiving if voice command system is accepting voice input or not

5.5. Application interface details

5.5.1. setSnr

- Function

  Set parameters used for the voice recognition start conditions. Set the recognition start conditions according to the amount (S/N ratio) of voice (noise) being input in the usage environment.

- Format

```
boolean setSnr (float snr) throws RemoteException;
```

- Parameter

  Value of the S/N ratio to be set. Minimum: 0, Maximum: 255.99, default: 10.0
  If the value is too large, the app should not be influenced by surrounding noise, however the user will need to talk loudly to trigger a response.

  For recommended value, refer to 5.8.2 Recommended value of voice recognition starting condition.

- Return value

Model	Explanation
boolean	Success: true, Failure: false

5.5.2. getSnr

- Function

  Acquire the set S/N ratio.

- Format

```
float getSnr() throws RemoteException;
```

- Parameter

  None.

- Return value

Model	Explanation
float	The current S/N ratio value used as the recognition start condition.

5.5.3. setAmp

- Function

  Set parameters used for the voice recognition start conditions. Set the amplitude value as one of the recognition start conditions.

- Format

```
boolean set Amp (int amp) throws RemoteException;
```

- Parameter

  Value of the amplitude to be set. Minimum: 0, Maximum: 32767, default: 1024
  If the value is too large, the app should not be influenced by surrounding noise, however the user will need to talk loudly to trigger a response.

  For recommended value, refer to 5.8.2 Recommended value of voice recognition starting condition.

- Return value

Model	Explanation
boolean	Success: true, Failure: false

5.5.4. getAmp

- Function

  Acquire the set amplitude value.

- Format

```
int getAmp() throws RemoteException;
```

- Parameter

  None.

- Return value

Model	Explanation
Int	Receive the value for the amplitude as the input voice recognition start condition.

5.5.5. start

- Function

  Enable voice input status.

- Format

```
boolean start(String path) throws RemoteException;
```

- Parameter

  Absolute path to the lms file. When null is specified, the lms file stored in the system is automatically selected and set (the default lms file will be used) based on the region information for the operating system of the BT-2000.

- Return value

Model	Explanation
boolean	Success: true, Failure: false

5.5.6. stop

- Function

  Disable voice input.

- Format

```
boolean stop(void) throws RemoteException;
```

- Parameter

  None.

- Return value

Model	Explanation
boolean	Success: true, Failure: false

5.5.7. registerCallback

- Function

  Register a callback to acquire the recognition results for a voice command.

- Format

```
void registerCallback(VoiceCommandClientCallbacks cb)
```

- Parameter

  VoiceCommandClientCallbacks class object

- Return value

  None

5.5.8. unRegisterCallback

- Function

  Cancel a callback to acquire the recognition results for a voice command.

- Format

```
void unRegisterCallback()
```

- Parameter

  None.

- Return value

  None.

5.5.9. getVoiceCommandSystemInterface

- Function

  Acquire a class as the interface to control voice commands.

- Format

```
IVoiceCommandInterface getVoiceCommandSystemInterface()
```

- Parameter

  None.

- Return value

Model	Explanation
IVoiceCommandInterface	Interface for voice command control

- Cautions

  Execute inside or after VoiceCommandClientCallbacks.onServiceConnected().

5.5.10. bindToRemoteRunningService

- Function

  Connect to a service class that provides voice command functions.

- Format

```
void bindToRemoteRunningService(Activity activity);
```

- Parameter

  An Activity object connected to the voice command service.

- Return value

  None.

5.5.11. unBindFromRemoteRunningService

- Function

  Disconnect from a voice command service.

- Format

```
void unBindFromRemoteRunningService(Activity activity)
```

- Parameter

  An Activity object connected to the voice command service.

- Return value

  None.

5.5.12.onVoiceCommand

- Function

  When detecting word that voice command system detected, callback returning ID of the word and character string.

- Format

```
void onVoiceCommand(int id, String word)
```

- Parameter

Model	Explanation
int id	ID number of the detected word
String word	Character string of the detected word

- Return value

  None

5.5.13.onSpeakable

- Function

  Callback noticed when voice command system voice receiving situation has changed.

- Format

```
void onSpeakable(boolean on)
```

- Parameter

Model	Explanation
boolean on	true: Voice input acceptable situation　false: Voice input not acceptable situation

- Return value

  None

5.6. Sample code

```
package com.epson.moverio.bt2pro.sample.vcmd;
import android.media.epson.IVoiceCommandInterface;
import android.media.epson.VoiceCommandClient;
import android.media.epson.VoiceCommandClientCallbacks;
import android.os.Bundle;
import android.os.RemoteException;
import android.app.Activity;
import android.content.Context;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity implements VoiceCommandClientCallbacks {
    private Context mContext = null;
    private Button mStartButton = null;
    private Button mStopButton = null;
    private VoiceCommandClient mVoiceCommandClient = null;
    private IVoiceCommandInterface mVoiceCommandSystem;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mContext = getApplicationContext();

        mVoiceCommandClient = new VoiceCommandClient();
        mVoiceCommandClient.bindToRemoteRunningService(this);

        mStartButton = (Button) findViewById(R.id.start);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    mVoiceCommandSystem.start("/system/vendor/bin/epe_au01_j1.lms");
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }
        });

        mStopButton = (Button) findViewById(R.id.stop);
        mStopButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                try {
                    mVoiceCommandSystem.stop();
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }
        });
    }
    @Override
    protected void onResume() {
        super.onResume();
        mVoiceCommandClient.registerCallback(this);
    }
    @Override
    protected void onPause() {
```

```
            super.onPause();
            mVoiceCommandClient.unRegisterCallback();
        }
            @Override
        protected void onDestroy() {
            super.onDestroy();
            mVoiceCommandClient.unBindFromRemoteRunningService(this);
        }
        @Override
        public void onServiceConnected() {
            mVoiceCommandSystem = mVoiceCommandClient.getVoiceCommandSystemInterface();
        }
        @Override
        public void onServiceDisconnected() {
        }
        @Override
        public void onVoiceCommand(int id, String word) {
            Toast.makeText(MainActivity.this, "MainActivity ID:"+id+"
                                    "+word, Toast.LENGTH_SHORT).show();
        }
        @Override
        public void onSpeakable(boolean speakable){
            Toast.makeText(MainActivity.this, "RA: " + speakable, Toast.LENGTH_SHORT).show();
        }
}
```

5.7. Default Supported Words

5.7.1. Voice information files (lms file) list and vocabulary list

- /system/vendor/bin/epe_au01_E1.lms default lms file（For English）

- /system/vendor/bin/epe_au01_E2.lms

・epe_au01_E1.lms

IDnumber	Voice input
2	Menu
3	Next
4	Return
5	Enter
6	Start
7	Stop
18	Display On

IDnumber	Voice input
19	Display Off
21	Blue
22	Red
23	Green
24	Yellow
31	Action1
32	Action 2

IDnumber	Voice input
33	Action 3
34	Action 4
35	Action 5
51	Zoom In
52	Zoom Out
101	Version

・epe_au01_E2.lms

IDnumber	Voice input
1	Home
8	Try again
9	Finish
10	Connect
11	Shoot
12	Rotate
13	Setting
14	Go Right

IDnumber	Voice input
15	Go Left
16	Go Up
17	Go Down
21	Blue
22	Red
23	Green
24	Yellow
41	Item1

IDnumber	Voice input
42	Item 2
43	Item 3
44	Item 4
45	Item 5
101	Version

5.8. Others

5.8.1. Timing for reflecting the setting for the voice recognition start conditions

When audio input recognition switches from disabled to enabled, the voice recognition start conditions are reflected in the system.

Therefore, if you change the recognition start conditions when audio input is enabled, once call IVoiceCommandInterface.stop(), disable audio input and call IVoiceCommandInterface.setSnr() to set the voice recognition starting condition. After IVoiceCommandInterface.start() is called, setting is then reflected by the system.

5.8.2. Recommended value of voice recognition starting condition

When recognition errors occur often due to noise of using environment, recognition errors can be reduced by changing the voice recognition starting condition according to the noisiness（noise level）.

Recommended value for each noise level is as below. This parameter is a setting value to have better voice recognition under white noise or machine noise. Optimum value is different by the noise type, so please set according to the actual used environment. For setting method, refer to 5.4 Application interface list.

Noise level(dB)	Noisiness	S/N ratio	Amplitude
35dB	Silent	10	1000
55dB	Little noisy	15	1500
70dB	Noisy	15	2500

For default value, refer to 5.5.1 setSnr and 5.5.3 setAmp.

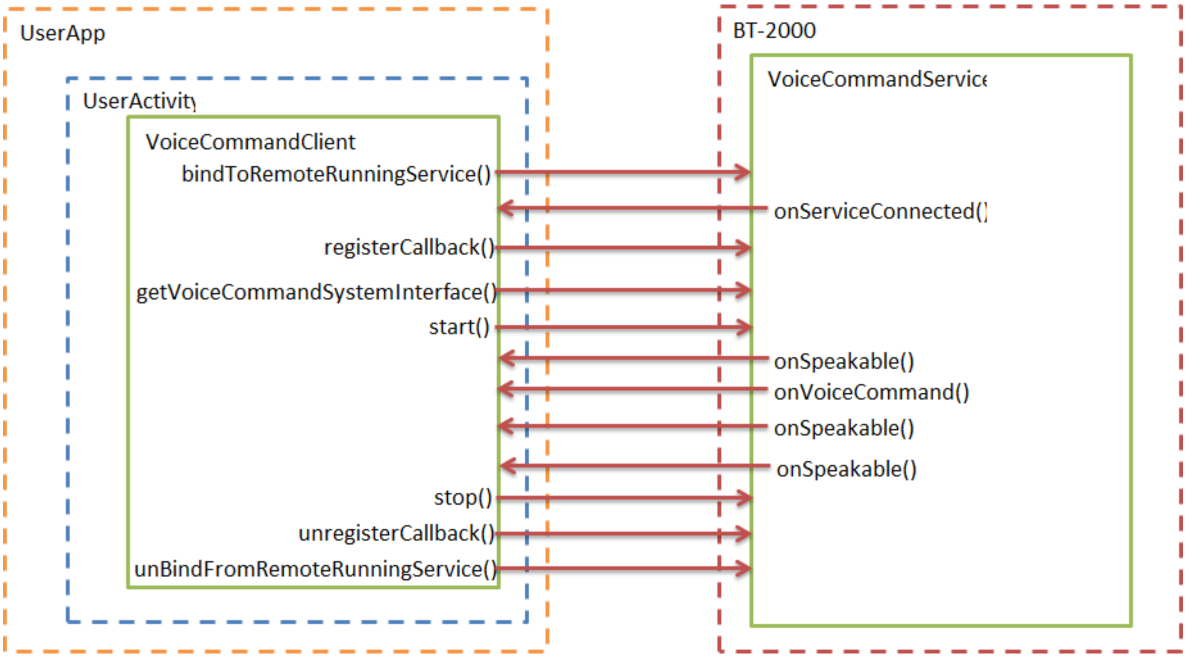
	S/N ratio	Amplitude
Default value	10	1024

5.8.3. Running voice command and video recording together

Due to Android 4.0 specification, voice command and video recording cannot run together.

For video recording, please finish voice command.

5.8.4. Voice command API usage flow example



See the sample source for voice commands for more details.

5.8.5. Callback onSpeakable for receiving speaking timing

In voice command system, when noise is inputted and judged as speech from S/N ratio or Amplitude value (Amp value), it enters into recognition process, and will not accept voice input for a moment. Also after recognizable word is detected, it will be in transferring process to be in recognition acceptable situation again, so there is a period that cannot accept voice input. By showing this period as GUI, it can improve voice command convenience. In details, , implement VoiceCommandClientCallbacks interface of

Android.media.epson.VoiceCommandClientCallbacksand define onSpeakable(), then can receive speech acceptable timing.

■Application to set S/N ratio and amplitude(AMP value) accorded to the environment

If onSpeakable is called when there is no speech situation, three is a possibility of reacting to surround noise. Therefore, using onSpeakble receiving value as index, set S/N ratio and Amp value to make value received from onSpeakable becomes always true while not speaking. By this way, it is possible to set S/N, amplitude (AMP value) more suitable for using environment.

5.8.6. Sample App

This chapter explains about voice command sample App that is pre-installed inBT-2000.

1. App information

App name : VoiceCommandSample.apk

Icon image

2. Summary

BT-2000 has function to recognize voice inputted from voice input equipment like headset microphone included, judge applicable word exist or not inside registered voice information file(lms), and notice ID and character string to App according to the judgment. This App is a sample App using this voce command function.

3． Function

ⅰ． Voice information（lms）file change

BT-2000 has voice information file written in 5.7.1Voice information files (lms file) list and vocabulary list and vocabulary list inside the system.By specifying these, it is available to detect registered word of selected file from inputted voice.f

ⅱ． Voice detection, receiving detected word and ID by App

When voice is inputted by microphone and applicable voice is detected, ID number related to the detected word will be noticed to the App. App will display ID and character string of the detected word to the display by toast.

ⅲ． Voice command parameter setting

Voice command function has a threshold as a parameter to decide whether inputted voice is am speech by human or a noise. Parameter has 2 series, S/N ratio and AMP value. By adjusting these, it is available to use voice command function where noise level is high.

・S/N ratio setting

Compared to surround noise, how much bigger sound inputted to be decided as a speech（Unit dB）．

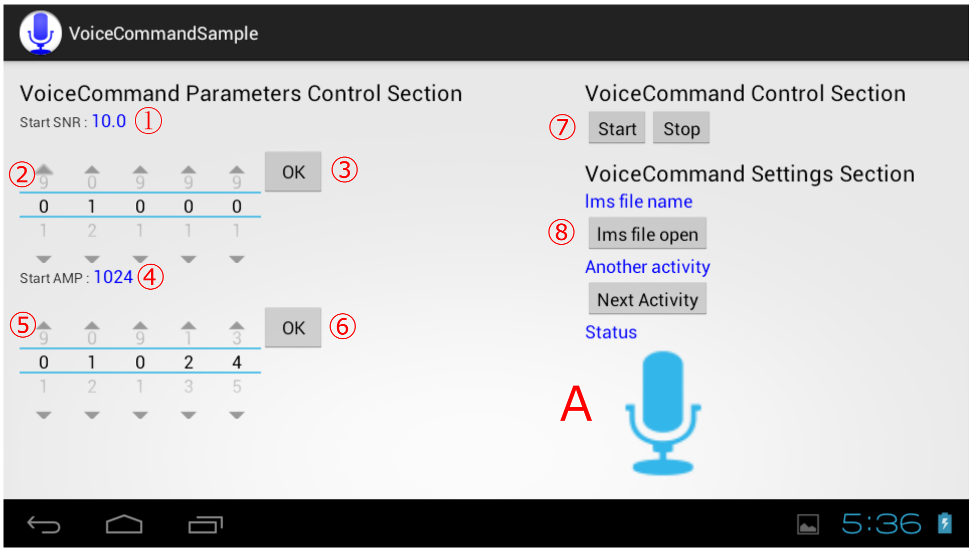
Min value 0 Max value 255.99 Initial setting 10.0

・AMP value setting

Sound pressure level of inputted voice to be decided as a speech.

Min value 0 Max value 32767 Initial setting 1024

4． Sample App operation method



① S/N ratio setting

Display S/N ratio setting value that is currently applied.

② S/N ratio changing UI

Can set value by selecting up arrow or down arrow or the value

Left 3 figures are integer figure, right 2 figures are decimal figure.

③ OK button（For S/N ratio setting）

Set the value selected by ② to the voice command system.

④ AMP setting value

Display AMP value that is currently applied.

⑤ AMP setting value changing UI

Can set value by selecting up arrow or down arrow or the value

⑥ OK button（For AMP value setting）

Set the value selected by ⑤ to the voice command system.

⑦ Start button, Stop button

Start button : Apply S/N ratio value, AMP value , voice information（Ims） to the voice command system and make voice detection effective.

Stop button : Invalid voice detection.

⑧ Display voice information（Ims）file

Display voice information (Ims) file that currently selected.

When App started, it displays ”Ims file name”. In this situation, when you make ⑦ voice detection effective, it operates as default voice information (Ims) hold inside system as the word recognition list.

⑨ Voice information（Ims） file changing file dialog button

Display file dialog for selecting voice information（Ims）file.

A GUI to display situation whether the voice command system accept voice input or not.
When blue microphone is displayed, voice input is acceptable. When red microphone is displayed, voice input is not acceptable.

・Remarks

At timing of Start button pressed, data necessary for system operation like S/N ratio and lms file will be applied to the system. While voice command is in operation（When voice detection is available）, setting value like S/N ratio that changed by the App does not apply when pressing the Start button. In this situation, press Stop button once, stop the voice command and then change again the setting value and press the Start button.

５.　　Voice information（lms）file

Use Voice information　（lms）　that　is stored inside BT-2000 system. For detail, refer to chapter 5.7.1Voice information files (lms file) list and vocabulary list.