PowerA MOGA API

Version	1.3.0
Date	16 January 2013

1 SUMMARY

This document describes the game controller library that will be available to Android developers who wish to use the PowerA MOGA and/or PowerA MOGA Pro in their applications.

One aim of the game controller library is simplicity of use; therefore the API is designed to be as minimal as possible.

Questions: Contact devsupport@PowerA.com

2 CONTENTS

3 HISTORY

Version	Date	Author	Comments
1.0.0	09 Feb 2012	JM	Initial version.
1.2.4	04 Apr 2012	JM	Added “Adding NDK Files” section. Added “Building NDK Project From Command Line” section. Added “Building NDK Project From Eclipse” section
1.2.6c	24 Apr 2012	JM	Added “Screen Management” section.
1.2.6i	25 May 2012	JM	Changed references to “marshalling” to a more correct “thread association” in “Listening Mode” section.
1.2.6o	02 Jul 2012	JM	Updated “Project Configuration” section.
1.2.7a	31 Jul 2012	JM	Added key, motion and state constants for future use in “Library Reference” section.
1.2.7b	03 Aug 2012	JM	Updated instructions in “Device Configuration” section.
1.2.7c	31 Aug 2012	JM	Added key code value-compatibility issue between MOGA and Android in “Library Reference” section.
1.2.7d	11 Sep 2012	JM	Added “MOGA Pivot” role in “Bluetooth Connectivity” and “Bluetooth Pairing” sections.
1.2.7e	04 Oct 2012	JM	Fixed documentation bugs in “Library Reference” section.
1.2.7i	20 Dec 2012	IB	Adding connection management documentation.
1.2.7j	16 Jan 2013	IB	Added developer configuration file functionality.
1.3.0	23 Jan 2012	IB & MD	MOGA Pro examples added. Connection config. Added Controller differences section.

4 CONFIGURATION

4.1 DEVICE CONFIGURATON

Install the service application on target Android device.

1. Remove any previous “BDA Controller Service” application from target Android device.
2. Remove any previous “MOGA Pivot” application from target Android device.
3. Install the “moga.pivot.sdk.apk” application on target Android device.

4.1.1 CREATING A NEW CONFIG FILE

Developers can enable additional debugging messages. This is activated by placing a configuration file on the target device.

This is useful for developers to gain more knowledge about how the MOGA controller service works in relation to their app. For example, this will allow developers to see when the app is calling onPause() and onResume(). This is important as if an app doesn’t handle these correctly it could result in a disconnection during gameplay.

1. On the target device.
    a. Create a new file called “mogaconfig.xml”, located at the root of the SD card. (Refer to “Environment.getExternalStorageDirectory()”)
2. Edit the configuration file (see below for an example).
    a. Open the configuration file.
    b. Add “<?xml version="1.0" encoding="utf-8"?>” to the configuration file.
    c. Add “<controller-service> </controller-service>” as the root element.
    d. Add zero or more “<property name="*name*" value="*value*" />” as a child to the < controller-service > element. See the available properties in 4.1.2.

```
<?xml version="1.0" encoding="utf-8"?>
<controller-service>
        <property name="debug.dev" value="true" />
</controller-service>
```

    e. Save and close this file.
3. Restart your device.

4.1.2 PROPERTY NAMES AND THEIR EFFECTS

The following table shows what each possible property in the configuration file is; what its valid range is and what it does.

Property Names	Property value range	Effect
debug.dev	True or False	Shows logs in logcat of when the following are called: • onPause() • onResume() • onScreenOff()

		• onScreenOn()

4.2 PROJECT CONFIGURATON

It is assumed that all developers are using an up-to-date installation of the developer tools. For reference, the game controller library and sample projects are built using:

- JDK 1.6 Update 27.
- Eclipse 3.7.2
- Android SDK r20 with API 8 & 10.
- Android Developer Tool (ADT) r20.
- Android NDK r7c.
- Unity 3.5.2f2.

4.2.1 ADDING LIBRARY FILES

To use the game controller library, the library file must be added to Android project.

In Eclipse with ADT:

1. Create the library folder.
    a. In Eclipse Package Explorer
        i. Right-click target project.
        ii. Select “New”.
        iii. Select “Folder”.
    b. In “New Folder” dialog
        i. Set “Folder name” to “libs”.
        ii. Click “Finish”.
2. Copy the library files.
    a. Copy “com.bda.controller.jar” file to the “libs” folder.
3. Configure the project. *This step seems to be optional since ADT r17.*
    a. In Eclipse Package Explorer
        i. Right click “com.bda.controller.jar” file.
        ii. Select “Build Path”.
        iii. Select “Add to Build Path”.
        iv. Confirm that “Referenced Libraries” folder has been created.
        v. Confirm that “com.bda.controller.jar” is a child of “Referenced Libraries” folder.

4.2.2 ADDING NDK FILES

To use the NDK, native source files must be added to Android project with Eclipse IDE and ADT.

It is assumed that any developer using native code is already familiar with the Android NDK and its build system. The following procedure summarizes the steps outlined in the Android NDK documentation. For

further information see http://developer.android.com/sdk/ndk/index.html. This procedure applies to both JNI and NativeActivity based projects.

In Eclipse with ADT:

1. Follow the steps outlined in “Adding Library File” above.
2. Create the JNI folder.
    a. In Eclipse Package Explorer
        i. Right-click target project.
        ii. Select “New”.
        iii. Select “Folder”.
    b. In “New Folder” dialog
        i. Set “Folder name” to “jni”.
        ii. Click “Finish”.
3. Copy the native files.
    a. Copy the native source files to the “jni” folder.
    b. Copy the “Android.mk” file to the 'jni' folder.
    c. Optionally copy the “Application.mk” file to the 'jni' folder.

4.2.3 BUILDING NDK PROJECT FROM COMMAND LINE

1. Building the JNI module.
    a. In command prompt
        i. Navigate to the target project directory.
        ii. Run the “ndk-build” script file (Linux) or “ndk-build.cmd” batch file (Windows).
2. Building the project.
    a. In Eclipse Package Explorer
        i. Right-click target project.
        ii. Select “Refresh”.
        iii. Build the target project.

4.2.4 BUILDING NDK PROJECT FROM ECLIPSE

1. Configure the project.
    a. In Eclipse Package Explorer
        i. Right-click target project
        ii. Select “Properties”.
    b. In “Properties” dialog
        i. Select “Builders”.
        ii. Click “New”
    c. In “Choose configuration type” dialog
        i. Select “Program”.
        ii. Click “OK”.
    d. In “Edit configuration” dialog
        i. Set “Name” to “NDK Builder” or similar.
        ii. Set “Location” to the path of the “ndk-build” script file (Linux) or “ndk-build.cmd” batch file (Windows).
        iii. Set “Working directory” to the path of the target project directory.
        iv. Optionally set “Refresh” and “Build Option” settings to suit.

v. Click “OK”.

e. In “Properties” dialog

i. Select “Builders”.

ii. Select “NDK Builder”.

iii. Click “Up” or “Down” to move it above “Android Package Builder”.

iv. Click “OK”.

2. Building the project.

a. Right-click target project in Eclipse Package Explorer.

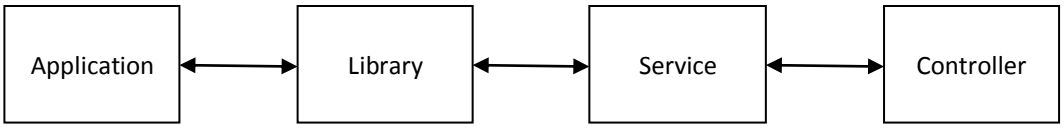
b. Select “Refresh”.

c. Build the target project.

5 LIBRARY OVERVIEW

The game controller library consists of a service application and a library file.

In order to use the game controller, the Android service application must be installed on the target Android device, and the library file must be used by the host activity.



5.1 BLUETOOTH CONNECTIVITY

The game controller library requires Bluetooth. The game controller library will monitor the Bluetooth state and adjust accordingly.

The game controller library will not handle enabling Bluetooth. This is left as a task for the game launcher and/or game application. The official game launcher, Moga Pivot, has support for enabling Bluetooth.

5.2 BLUETOOTH PAIRING

The game controller library will only connect to currently paired devices. The game controller library will monitor the Bluetooth pairing state and adjust accordingly.

The game controller library will not handle pairing of Bluetooth devices. This is left as a task for the game launcher and/or game application. The official game launcher, Moga Pivot, has support for pairing of Bluetooth devices.

5.3 SCREEN MANAGEMENT

The game controller library does not prevent the screen from dimming or turning off. It is the developer's responsibility to manage the screen to suit the need of their application. If the developer needs to keep the screen on, then the android.view.View.setKeepScreenOn() method or android.os.PowerManager API may be of interest.

5.4 CONNECTION MANAGEMENT

By default, while the service is not connected to a game controller, it will continually try to connect in the background. However these background connection attempts affect other Bluetooth and Wi-Fi connections and often introduce 'lag'.

Developers should restore allowing of new connections on shutdown. It is recommended developers avoid using the provided functions unless absolutely necessary. See 7.1.2.9 to 7.1.2.12.

6 LIBRARY USAGE

The entry point to the game controller library is the Controller object.

It is highly recommended that each Activity, that requires access to the game controller, should have its own Controller object.

6.1 INITIALIZATION AND SHUTDOWN

The host activity obtains a Controller object and initializes it on creation. The host activity shuts down the Controller object on destruction.

When the Controller object is initialized, it will attempt to connect with the service application. The service connection is performed asynchronously and therefore there is a delay before the library returns valid data. It is recommended not to query the controller during this period. Continuous polling or using the listener will bypass this issue.

The following sample code demonstrates the game controller library usage.

```
package com.example.game;

import com.bda.controller.Controller;

public class GameActivity extends Activity
{
    Controller mController = null;

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

        mController = Controller.getInstance(this);
        mController.init();
    }

    @Override
    protected void onDestroy()
    {
        if(mController != null)
        {
            mController.exit();
        }
        super.onDestroy();
    }
}
```

6.2 RESUMING AND PAUSING

The host activity must notify the Controller object when it is paused and when it has resumed. This allows the Controller object to change its behavior accordingly.

```
public class GameActivity extends Activity
{
    ...

    @Override
    protected void onPause()
    {
        super.onPause();
        if(mController != null)
        {
            mController.onPause();
        }
    }

    @Override
    protected void onResume()
    {
        super.onResume();
        if(mController != null)
        {
            mController.onResume();
        }
    }

    ...
}
```

6.3 CONTROLLER ACCESS

There are two ways to obtain the state of the game controller. The host activity may use one or both methods to monitor the game controller.

6.3.1 POLLING MODE

Polling mode is designed for developers who prefer the traditional approach to game input. The host activity can directly query the Controller object for the current state of the game controller. Polling mode is the recommended way to access the controller.

For demonstration purposes, the onGameUpdate() method is a proxy for the host activity's update method.

```
public class GameActivity extends Activity
{
    ...

    void onGameUpdate()
    {
        if(mController != null)
        {
            // test if controller is connected
            if(mController.getState(Controller.STATE_CONNECTION) == Controller.ACTION_CONNECTED)
            {
                // test if button a is pressed
                if(mController.getKeyCode(Controller.KEYCODE_BUTTON_A) == Controller.ACTION_DOWN)
                {
                    // button A is pressed
                }
                else
                {
                    // button A is released
                }

                // test if d-pad left is pressed
                if(mController.getKeyCode(Controller.KEYCODE_DPAD_LEFT) == Controller.ACTION_DOWN)
                {
                    // d-pad left is pressed
                }
                else
                {
                    // d-pad left is released
                }

                // read left analog stick
                mX = mController.getAxisValue(Controller.AXIS_X);
                mY = mController.getAxisValue(Controller.AXIS_Y);

                // test if controller is in low power state
                if(mController.getState(Controller.STATE_POWER_LOW) == Controller.ACTION_TRUE)
                {
                    // low power state
                }
                else
                {
                    // normal power state
                }
            }
        }
    }

    ...
}
```

6.3.2 LISTENING MODE

Listening mode is designed for developers who prefer the Android approach to input. The host activity can register a ControllerListener to the Controller object to listen for events from the game controller. This approach is recommended only for advanced developers, as there are considerations to be aware of.

- *Past events.* The game controller library does not resend past events. For example, if the game controller is already connected to the game controller library before the host activity has registered a ControllerListener, then the host activity will not be informed of connection state as the event has already occured in the past. The recommended solution is to perform an one-shot poll of state, key and axis values that the host activity is interested in.
- *Pause and resume.* The game controller library suspends dispatching of KeyEvent and MotionEvents to the ControllerListener when the activity is “paused”. This prevents background activities from inadvertently responding to the game controller. As a consequence, a resumed activity’s idea of the

controller state may not match reality due to missed events. The recommended solution is to perform an one-shot poll of state, key and axis values that the host activity is interested in, upon resumption.

- *Thread association.* The game controller library can be configured to call the ControllerListener's methods with or without a specified Handler. If a Handler is not supplied when the ControllerListener is registered, then the ControllerListener's methods will be called in arbitary non-UI thread. If a Handler is supplied when the ControllerListener is registered, then the ControllerListener's method will be called in the thread associated with the Handler. Whether a Handler is required depends on the situation, typically a Handler is required to deal with some restriction or synchronization issue. For example, most of the Android UI API is restricted to the UI thread, therefore the ControllerListener's methods can use the API if the ControllerListener was specified to execute in the UI thread.

For demonstration purposes, the ControllerListener is implemented by the GameActivity object. Developers may prefer to use alternative methods of implementing ControllerListener.

```
public class GameActivity extends Activity implements ControllerListener
{
    ...

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

        mController = Controller.getInstance(this);
        mController.init();
        mController.setListener(this, new Handler());
    }

    public void onKeyEvent(KeyEvent event)
    {
        switch(event.getKeyCode())
        {
        case KeyEvent. KEYCODE_BUTTON_A:
            if(event.getAction() == KeyEvent.ACTION_DOWN)
            {
                // button A has been pressed
            }
            else
            {
                // button A has been released
            }
            break;

        case KeyEvent. KEYCODE_DPAD_LEFT:
            if(event.getAction() == KeyEvent.ACTION_DOWN)
            {
                // d-pad left has been pressed
            }
            else
            {
                // d-pad left has been released
            }
            break;
        }
    }

    public void onMotionEvent(MotionEvent event)
    {
        // read left analog stick
        mX = event.getAxisValue(MotionEvent.AXIS_X);
        mY = event.getAxisValue(MotionEvent.AXIS_Y);
    }

    public void onStateEvent(StateEvent event)
    {
        switch(event.getState())
        {
        case StateEvent.STATE_CONNECTION:
            switch(event.getAction())
            {
            case StateEvent.ACTION_DISCONNECTED:
                // disconnected from controller
                break;

            case StateEvent.ACTION_CONNECTED:
                // connected to controller
                break;

            case StateEvent.ACTION_CONNECTING:
                // attempting to connect to controller
                break;
            }
            break;

        case StateEvent.STATE_POWER_LOW:
            if(event.getAction() == StateEvent.ACTION_TRUE)
```

```
            {
                // controller has entered low power state
            }
            else
            {
                // controller has entered normal power state
            }
            break;
        }
    }
}
```

6.4 DETECTING CONTROLLER VERSION

6.4.1 GETTING THE CONTROLLER VERSION

This involves calling Controller.getState() using the "Controller.*STATE_CURRENT_PRODUCT_VERSION*" argument. If the return value is Controller.ACTION_VERSION_MOGA, then the connected game controller is a Moga Controller, if the return value is Controller.ACTION_VERSION_MOGAPRO, then the connected game controller is a Moga Pro Controller.

```
int getMogaControllerVersion()
{
    return mController.getState(Controller.STATE_CURRENT_PRODUCT_VERSION);
}
```

An example implementation can be seen in the LauncherActivity.getMogaControllerVersion() method.

6.4.2 DEALING WITH CONTROLLER DIFFERENCES

Moga controllers have a left analogue stick that doubles as a d-pad. Developers must be careful when developing a game using both the analogue stick and d-pad independently. It may be beneficial to have two separate controller profiles depending on the controller version. See the following example code:

```
class ExampleControllerListener implements ControllerListener {
        @Override
        public void onKeyEvent(KeyEvent event) {
                switch (mController.getState(Controller.STATE_CURRENT_PRODUCT_VERSION)) {
                case Controller.ACTION_VERSION_MOGA:
                        switch (event.getKeyCode()) {
                        case KeyEvent.KEYCODE_BUTTON_A:
                                mPlayer.mButtonA = event.getAction();
                                break;
                        case KeyEvent.KEYCODE_BUTTON_B:
                                mPlayer.mButtonB = event.getAction();
                                break;
                        case KeyEvent.KEYCODE_BUTTON_START:
                                mPlayer.mButtonStart = event.getAction();
                                break;
                        }
                        break;
                case Controller.ACTION_VERSION_MOGAPRO:
                        switch (event.getKeyCode()) {
                        case KeyEvent.KEYCODE_BUTTON_X:
                                mPlayer.mButtonA = event.getAction();
                                break;
                        case KeyEvent.KEYCODE_BUTTON_Y:
                                mPlayer.mButtonB = event.getAction();
                                break;
                        case KeyEvent.KEYCODE_BUTTON_START:
                                mPlayer.mButtonStart = event.getAction();
                                break;
                        }
                }
        }
}
```

6.4.3 TABLE OF DIFFERENCES

Feature	Moga Controller (Pocket)	Moga Pro Controller
Dpad and Left analogue Stick	Linked output.	Independent output.
Trigger buttons (L2, R2)	No	Yes
Thumb buttons (L3, R3)	No	Yes

7 LIBRARY REFERENCE

The game controller library is contained in the “com.bda.controller” package.

7.1 CLASS CONTROLLER

7.1.1 CONSTANTS

Where possible, the Controller constants are value-compatible with the equivalent constants in Android’s own MotionEvent and KeyEvent class, and therefore, can be used interchangeably.

In API 1.2.7b and earlier, there was a compatibility issue with the values of the KEYCODE_BUTTON_X and KEYCODE_BUTTON_Y constants in the game controller’s namespace and Android’s namespace. This has been corrected in API 1.2.7c and later. Developers should use the constants declared in game controller library’s Controller class for best results.

int	STATE_CONNECTION	State constant: connection state. ACTION_DISCONNECTED, ACTION_CONNECTED or ACTION_CONNECTING.
int	STATE_POWER_LOW	State constant: low power state. Returns ACTION_FALSE or ACTION_TRUE.
int	STATE_SUPPORTED_VERSION	State constant: supported protocol version. Returns the latest protocol version that can be used in a connection. *Since API 1.2.7a.* *Depreciated: Please use* STATE_SUPPORTED_PRODUCT_VERSION
int	STATE_SELECTED_VERSION	State constant: selected protocol version. Returns the current protocol version used in a connection. *Since API 1.2.7a.* *Depreciated: Please use* STATE_CURRENT_PRODUCT_VERSION
int	STATE_SUPPORTED_PRODUCT_VERSION	State constant: supported protocol version. Returns the latest protocol version that can be used

		in a connection. *Since API 1.3.0*
int	STATE_CURRENT_PRODUCT_VERSION	State constant: selected protocol version. Returns the current protocol version used in a connection. *Since API 1.3.0.*
int	ACTION_DISCONNECTED	getState() value: the state is disconnected. Compatible with ACTION_FALSE.
int	ACTION_CONNECTED	getState() value: the state is connected. Compatible with ACTION_TRUE.
int	ACTION_CONNECTING	getState() value: the state is connecting.
int	ACTION_VERSION_MOGA	getState() value: the Controller is a Moga.
int	ACTION_VERSION_MOGAPRO	getState() value: the Controller is a Moga Pro.

int	KEYCODE_BUTTON_A	Key code constant: A Button
int	KEYCODE_BUTTON_B	Key code constant: B Button
int	KEYCODE_BUTTON_X	Key code constant: X Button. *In API 1.2.7b and earlier, do not use android.view.KeyEvent.KEYCODE_BUTTON_X in place of this key code, due to value-compatibility issue.*
int	KEYCODE_BUTTON_Y	Key code constant: Y Button key. *In API 1.2.7b and earlier, do not use android.view.KeyEvent.KEYCODE_BUTTON_Y in place of this key code, due to value-compatibility issue.*
int	KEYCODE_BUTTON_START	Key code constant: Start Button.
int	KEYCODE_BUTTON_SELECT	Key code constant: Select Button.
int	KEYCODE_BUTTON_L1	Key code constant: L1 Button.
int	KEYCODE_BUTTON_R1	Key code constant: R1 Button.
int	KEYCODE_BUTTON_L2	Key code constant: L2 Button. *Since API 1.2.7a.*
int	KEYCODE_BUTTON_R2	Key code constant: R2 Button. *Since API 1.2.7a.*
int	KEYCODE_BUTTON_THUMBL	Key code constant: Left Thumb Button. *Since API 1.2.7a.*
int	KEYCODE_BUTTON_THUMBR	Key code constant: Right Thumb Button. *Since API 1.2.7a.*
int	KEYCODE_DPAD_UP	Key code constant: Directional Pad Up.
int	KEYCODE_DPAD_DOWN	Key code constant: Directional Pad Down.
int	KEYCODE_DPAD_LEFT	Key code constant: Directional Pad Left.
int	KEYCODE_DPAD_RIGHT	Key code constant: Directional Pad Right.
int	ACTION_DOWN	getKeyCode() value: The key has been pressed down.
int	ACTION_UP	getKeyCode() value: The key has been released.

int	AXIS_X	Axis constant: X axis of a motion event. For a joystick, reports the absolute X position of the joystick. The value is normalized to a range from -1.0 (left) to 1.0 (right).
int	AXIS_Y	Axis constant: Y axis of a motion event. For a joystick, reports the absolute Y position of the joystick. The value is normalized to a range from -1.0 (up or far) to 1.0 (down or near).
int	AXIS_Z	Axis constant: Z axis of a motion event. For a joystick, reports the absolute Z position of the joystick. The value is normalized to a range from -1.0 (high) to 1.0 (low). *On game pads with two analog joysticks, this axis is often reinterpreted to report the absolute X position of the second joystick instead*.
int	AXIS_RZ	Axis constant: Z Rotation axis of a motion event For a joystick, reports the absolute rotation angle about the Z axis. The value is normalized to a range from -1.0 (counter-clockwise) to 1.0 (clockwise). *On game pads with two analog joysticks, this axis is often reinterpreted to report the absolute Y position of the second joystick instead.*
int	AXIS_LTRIGGER	Axis constant: Left Trigger axis of a motion event. For a joystick, reports the absolute position of the left trigger control. The value is normalized to a range from 0.0 (released) to 1.0 (fully pressed). *Since API 1.2.7a.*
int	AXIS_RTRIGGER	Axis constant: Right Trigger axis of a motion event. For a joystick, reports the absolute position of the right trigger control. The value is normalized to a range from 0.0 (released) to 1.0 (fully pressed). *Since API 1.2.7a.*

int	INFO_KNOWN_DEVICE_COUNT	Information constant: The number of known (paired) devices. This is only valid when Bluetooth is enabled
int	INFO_ACTIVE_DEVICE_COUNT	Information constant: The number of active (connected) devices. This is only valid when Bluetooth is enabled.

7.1.2 METHODS

7.1.2.1 GETINSTANCE

static Controller getInstance(Context context)

Description

Gets a new instance of a game controller library object.

Parameters

Context	Context	

Return Value

Returns a *Controller* object.

7.1.2.2 INIT

boolean init()

Description

Initializes the game controller library.

Parameters

None

Return Value

Return true if successful, false otherwise.

Notes

Refer to section "Initialization and Shutdown" for additional notes.

7.1.2.3 EXIT

void exit()

Description

Un-initializes the game controller library.

Parameters

None

Return Value

None

Notes

Refer to section "Initialization and Shutdown" for additional notes.

7.1.2.4 ONRESUME

void onResume()

Description

Notifies the game controller library that the host activity has been resumed.

Parameters

None

Return Value

None

Notes

Refer to section "Resuming and Pausing" for additional notes.

7.1.2.5 ONPAUSE

void onPause()

Description

Notifies the game controller library that the host activity has been paused.

Parameters

None

Return Value

None

Notes

Refer to section "Resuming and Pausing" for additional notes.

7.1.2.6 GETAXISVALUE

float getAxisValue(int axis)

Description

Returns the value of the requested axis.

Parameters

int	axis	The axis identifier for the axis value to retrieve. Any of the AXIS_xxx constants.

Return Value

The value of the axis, or 0 if the axis is not available.

7.1.2.7 GETKEYCODE

int getKeyCode(int keyCode)

Description

Retrieve the value of the requested key code.

Parameters

int	keyCode	The key code identifier for the key code value to retrieve. Any of the KEYCODE_xxx constants.

Return Value

The event action: ACTION_DOWN or ACTION_UP.

7.1.2.8 GETINFO

int getInfo()

Description

Gets information about the game controller library.

Parameters

int	info	Any of the INFO_xxx constants.

Return Value

Returns the requested information value.

7.1.2.9 ALLOWNEWCONNECTIONS

void allowNewConnections()

Description

Allows new connections to be made to unconnected game controllers.

Parameters

None.

Return Value

None.

Notes

By default, while the service is not connected to a game controller, it will continually try to connect in the background. However these background connection attempts affect other Bluetooth and Wi-Fi connections and often introduce 'lag'. This method allows the developer to temporarily enable background connections as required.

Developers should restore allowing of new connections on shutdown, due to the possibility of a rogue disallowNewConnections() halting the service. It is recommended that developers avoid using this function.

7.1.2.10 DISALLOWNEWCONNECTIONS

void disallowNewConnections()

Description

Prevents new connections to be made to unconnected game controllers.

Parameters

None.

Return Value

None.

Notes

By default, while the service is not connected to a game controller, it will continually try to connect in the background. However these background connection attempts affect other Bluetooth and Wi-Fi connections and often introduce 'lag'. This method allows the developer to temporarily disable background connections as required.

Developers should restore allowing of new connections on shutdown, due to the possibility of a rogue disallowNewConnections() halting the service. It is recommended that developers avoid using this function.

7.1.2.11 ISALLOWINGNEWCONNECTIONS

void isAllowingNewConnections()

Description

Allows new connections to be made to the service.

Parameters

None.

Return Value

The Boolean value indicating whether new connections are allowed.

7.1.2.12 GETSTATE

int getState(int state)

Description

Retrieve the value of the requested state.

Parameters

int	state	The state identifier for the state to retrieve. Any of the STATE_xxx constants.

Return Value

The event action: ACTION_FALSE or ACTION_TRUE. Some states may return other values.

7.1.2.13 SETLISTENER

void setListener(ControllerListener listener, Handler handler)

Description

Set an event listener for the game controller library.

Parameters

ControllerListener	listener	Event listener to use. If *null*, clears the previous listener. If non-*null*, replaces the previous listener.
Handler	handler	Handler to use for marshalling. If *null*, the *listener* will be called in a non-UI thread. If non-*null*, the *listener* will be called in the thread associated with the handler. The handler is only used if *listener* is non-*null*.

Return Value

None

Notes

Refer to section "Listening mode" for additional notes.

7.2 INTERFACE CONTROLLERLISTENER

7.2.1 METHODS

7.2.1.1 ONKEYEVENT

void onKeyEvent(KeyEvent event)

Description

Called when a button or direction-pad event has occurred.

Parameters

KeyEvent	event	The object that describes the event.

Return Value

None

7.2.1.2 ONMOTIONEVENT

void onMotionEvent(MotionEvent event)

Description

Called when an analogue button or joystick event has occurred.

Parameters

MotionEvent	event	The object that describes the event.

Return Value

None

7.2.1.3 ONSTATEEVENT

void onStateEvent(StateEvent event)

Description

Called when a state event has occurred.

Parameters

StateEvent	event	An object describing the key event.

Return Value

None

7.3 CLASS KEYEVENT

The KeyEvent class is modeled after Android's own KeyEvent class.

7.3.1 CONSTANTS

Where possible, the KeyEvent constants are value-compatible with the equivalent constants in Android's own KeyEvent class, and therefore, can be used interchangeably.

In API 1.2.7b and earlier, there was a compatibility issue with the values of the KEYCODE_BUTTON_X and KEYCODE_BUTTON_Y constants in the game controller's namespace and Android's namespace. This has been corrected in API 1.2.7c and later. Developers should use the constants declared in game controller library's KeyEvent class for best results.

int	KEYCODE_BUTTON_A	Key code constant: A Button key
int	KEYCODE_BUTTON_B	Key code constant: B Button key
int	KEYCODE_BUTTON_X	Key code constant: X Button key.

		In API 1.2.7b and earlier, do not use android.view.KeyEvent.KEYCODE_BUTTON_X in place of this key code, due to value-compatibility issue.
int	KEYCODE_BUTTON_Y	Key code constant: Y Button key. *In API 1.2.7b and earlier, do not use android.view.KeyEvent.KEYCODE_BUTTON_Y in place of this key code, due to value-compatibility issue.*
int	KEYCODE_BUTTON_START	Key code constant: Start Button key.
int	KEYCODE_BUTTON_SELECT	Key code constant: Select Button key.
int	KEYCODE_BUTTON_L1	Key code constant: L1 Button key.
int	KEYCODE_BUTTON_R1	Key code constant: R1 Button key.
int	KEYCODE_BUTTON_L2	Key code constant: L2 Button key. *Since API 1.2.7a.*
int	KEYCODE_BUTTON_R2	Key code constant: R2 Button key. *Since API 1.2.7a.*
int	KEYCODE_BUTTON_THUMBL	Key code constant: Left Thumb Button key. *Since API 1.2.7a.*
int	KEYCODE_BUTTON_THUMBR	Key code constant: Right Thumb Button key. *Since API 1.2.7a.*
int	KEYCODE_DPAD_UP	Key code constant: Directional Pad Up key.
int	KEYCODE_DPAD_DOWN	Key code constant: Directional Pad Down key.
int	KEYCODE_DPAD_LEFT	Key code constant: Directional Pad Left key.
int	KEYCODE_DPAD_RIGHT	Key code constant: Directional Pad Right key.
int	ACTION_DOWN	getKeyCode() value: the key has been pressed down.
int	ACTION_UP	getKeyCode() value: the key has been released.

7.3.2 METHODS

7.3.2.1 GETACTION

int getAction()

Description

Retrieve the action of this key event.

Parameters

None

Return Value

The event action: ACTION_DOWN or ACTION_UP.

7.3.2.2 GETEVENTTIME

long getEventTime()

Description

Retrieve the time this event occurred, in the uptimeMillis() time base.

Parameters

None

Return Value

Returns the time this event occurred, in the uptimeMillis() time base.

7.3.2.3 GETKEYCODE

int getKeyCode()

Description

Retrieve the key code of this key event.

Parameters

None

Return Value

The key code of the event. This is any of the KEYCODE_xxx constants.

7.4 CLASS MOTIONEVENT

The MotionEvent class is modeled after Android's own MotionEvent class.

7.4.1 CONSTANTS

Where possible, the MotionEvent constants are value-compatible with the equivalent constants in Android's own MotionEvent class, and therefore, can be used interchangeably.

int	AXIS_X	Axis constant: X axis of a motion event. For a joystick, reports the absolute X position of the joystick. The value is normalized to a range from -1.0 (left) to 1.0 (right).
int	AXIS_Y	Axis constant: Y axis of a motion event. For a joystick, reports the absolute Y position of the joystick. The value is normalized to a range from -1.0 (up or far) to 1.0 (down or near).
int	AXIS_Z	Axis constant: Z axis of a motion event. For a joystick, reports the absolute Z position of the joystick. The value is normalized to a range from -1.0 (high) to 1.0 (low). *On game pads with two analog joysticks, this axis is often reinterpreted to report the absolute X position*

		of the second joystick instead.
int	AXIS_RZ	Axis constant: Z Rotation axis of a motion event For a joystick, reports the absolute rotation angle about the Z axis. The value is normalized to a range from -1.0 (counter-clockwise) to 1.0 (clockwise). *On game pads with two analog joysticks, this axis is often reinterpreted to report the absolute Y position of the second joystick instead.*
int	AXIS_LTRIGGER	Axis constant: Left Trigger axis of a motion event. For a joystick, reports the absolute position of the left trigger control. The value is normalized to a range from 0.0 (released) to 1.0 (fully pressed). *Since API 1.2.7a.*
int	AXIS_RTRIGGER	Axis constant: Right Trigger axis of a motion event. For a joystick, reports the absolute position of the right trigger control. The value is normalized to a range from 0.0 (released) to 1.0 (fully pressed). *Since API 1.2.7a.*

7.4.2 METHODS

7.4.2.1 FINDPOINTERINDEX

int findPointerIndex (int pointerId)

Description

Given a pointer identifier, find the index of its data in the event.

Parameters

int	pointerId	The identifier of the pointer to be found.

Return Value

Returns either the index of the pointer (for use with getX(int) et al.), or -1 if there is no data available for that pointer identifier.

7.4.2.2 GETAXISVALUE

float getAxisValue(int axis)

Description

getAxisValue(int) for the first pointer index (may be an arbitrary pointer identifier).

Parameters

int	axis	The axis identifier for the axis value to retrieve. Any of the AXIS_xxx constants.

Return Value

The value of the axis, or 0 if the axis is not available.

7.4.2.3 GETAXISVALUE

float getAxisValue(int axis, int pointerIndex)

Description

Returns the value of the requested axis for the given pointer index (use getPointerId(int) to find the pointer identifier for this index).

Parameters

int	axis	The axis identifier for the axis value to retrieve. Any of the AXIS_xxx constants.
int	pointerIndex	Raw index of pointer to retrieve. Value may be from 0 (the first pointer that is down) to getPointerCount()-1.

Return Value

The value of the axis, or 0 if the axis is not available.

7.4.2.4 GETEVENTTIME

long getEventTime()

Description

Retrieve the time this event occurred, in the uptimeMillis() time base.

Parameters

None

Return Value

Returns the time this event occurred, in the uptimeMillis() time base.

7.4.2.5 GETPOINTERCOUNT

int getPointerCount()

Description

The number of pointers of data contained in this event.

Parameters

None

Return Value

The number of pointers of data contained in this event. Always >= 1.

7.4.2.6 GETPOINTERID

int getPointerId(int pointerIndex)

Description

Return the pointer identifier associated with a particular pointer data index is this event.

Parameters

int	pointerIndex	Raw index of pointer to retrieve. Value may be from 0 (the first pointer that is down) to getPointerCount()-1.

Return Value

Return the pointer identifier associated with a particular pointer data index is this event. The identifier tells you the actual pointer number associated with the data, accounting for individual pointers going up and down since the start of the current gesture.

7.4.2.7 GETRAWX

float getRawX()

Description

Returns the original raw X coordinate of this event.

Parameters

None

Return Value

Returns the original raw X coordinate.

7.4.2.8 GETRAWY

float getRawY()

Description

Returns the original raw Y coordinate of this event.

Parameters

None

Return Value

Returns the original raw Y coordinate.

7.4.2.9 GETX

float getX()

Description

getX(int) for the first pointer index (may be an arbitrary pointer identifier).

Parameters

None

Return Value

Returns the X coordinate.

7.4.2.10 GETX

float getX(int pointerIndex)

Description

Returns the X coordinate of this event for the given pointer index (use getPointerId(int) to find the pointer identifier for this index).

Parameters

int	pointerIndex	Raw index of pointer to retrieve. Value may be from 0 (the first pointer that is down) to getPointerCount()-1.

Return Value

Returns the X coordinate.

7.4.2.11 GETXPRECISION

float getXPrecision()

Description

Return the precision of the X coordinates being reported. You can multiply this number with getX() to find the actual hardware value of the X coordinate

Parameters

None

Return Value

Returns the precision of X coordinates being reported.

7.4.2.12 GETY

float getY()

Description

getY(int) for the first pointer index (may be an arbitrary pointer identifier).

Parameters

None

Return Value

Returns the Y coordinate.

7.4.2.13 GETY

float getY(int pointerIndex)

Description

Returns the Y coordinate of this event for the given pointer index (use getPointerId(int) to find the pointer identifier for this index).

Parameters

int	pointerIndex	Raw index of pointer to retrieve. Value may be from 0 (the first pointer that is down) to getPointerCount()-1.

Return Value

Returns the Y coordinate.

7.4.2.14 GETYPRECISION

float getYPrecision()

Description

Return the precision of the Y coordinates being reported. You can multiply this number with getY() to find the actual hardware value of the Y coordinate

Parameters

None

Return Value

Returns the precision of Y coordinates being reported.

7.5 CLASS STATEEVENT

7.5.1 CONSTANTS

int	STATE_CONNECTION	State constant: connection state. Returns ACTION_DISCONNECTED,

		ACTION_CONNECTED and ACTION_CONNECTING.
int	STATE_POWER_LOW	State constant: low power state. Returns ACTION_FALSE and ACTION_TRUE.
int	STATE_SUPPORTED_VERSION	State constant: supported protocol version. Returns the best protocol version that can be used in a connection. *Since API 1.2.7a.* *Depreciated: Please use* STATE_SUPPORTED_PRODUCT_VERSION
int	STATE_SELECTED_VERSION	State constant: selected protocol version. Returns the current protocol version used in a connection. *Since API 1.2.7a.* *Depreciated: Please use* STATE_CURRENT_PRODUCT_VERSION
int	STATE_SUPPORTED_PRODUCT_VERSION	State constant: supported protocol version. Returns the best protocol version that can be used in a connection. *Since API 1.3.0*
int	STATE_CURRENT_PRODUCT_VERSION	State constant: selected protocol version. Returns the current protocol version used in a connection. *Since API 1.3.0.*
int	ACTION_FALSE	getAction() value: the state is false.
int	ACTION_TRUE	getAction() value: the state is true.
int	ACTION_DISCONNECTED	getAction() value: the state is disconnected. Compatible with ACTION_FALSE.
int	ACTION_CONNECTED	getAction() value: the state is connected. Compatible with ACTION_TRUE.
int	ACTION_CONNECTING	getAction() value: the state is connecting.
int	ACTION_VERSION_MOGA	getAction() value: the Controller version is a Moga.
int	ACTION_VERSION_MOGAPRO	getAction() value: the Controller version is a Moga Pro.

7.5.2 METHODS

7.5.2.1 GETACTION

int getAction()

Description

Retrieve the action of this state event.

Parameters

None

Return Value

The event action: typically ACTION_FALSE or ACTION_TRUE. Some states may return more values.

7.5.2.2 GETEVENTTIME

long getEventTime()

Description

Retrieve the time this event occurred, in the uptimeMillis() time base.

Parameters

None

Return Value

Returns the time this event occurred, in the uptimeMillis() time base.

7.5.2.3 GETSTATE

int getState()

Description

Retrieve the state of this state event

Parameters

None

Return Value

The state of the event. This is any of the STATE_xxx constants.