阿里 TV 外设使用手册

版本号：V1.09

创建时间：2013-12-14

更新时间：2015-02-15

修改记录

编号	版本	内容	日期	修改人
1	0.50	创建文档	2013-12-14	王永
2	0.60	主要修改点： 1.去掉了 EXDeviceManager 中 setOnRawDataListener API，raw data 和通过其他 Event 同时传出； 2. 修改了 RawData 类，将 mDeviceMark 改为 mDeviceId，去掉 mRawDataLength； 3. 去掉了 EXDevice 中 getKeyList API； 4. 增加了 resetAMousePos API。	2013-12-23	王永
3	0.61	EXDevice 类增加了三个模拟系统事件的回写接口： sendSysKeyEvent, sendSysMouseEvent, sendSysSTouchEvent	2013-12-26	王永
4	0.62	EXDevice 类增加了一次性获取 Joystick 和 Sensor 数据的接口 getJMotionValues, getMSensorValues EXDeviceManager 类增加了设置 HostDeviceId 的接口 setHostDeviceId Global 类开放 Log 开关	2013-12-27	王永
5	0.63	EXDeviceManager 类增加了“系统事件模式”的开关接口： setEnableSysEventMode	2013-12-30	王永
6	0.64	EXDeviceManager 类增加通过 DeviceId 获取 EXDevice 对象的方法：EXDevice getDevice(int deviceId)	2014-1-3	王永
7	0.65	EXDeviceManager 类增加通过获取主设备 ID 的接口等 getHostDeviceId	2014-1-8	王永
8	0.70	增加 BaseActivity 类，该类作为 app 开发的父类，简化开发的代码开发量	2014-1-9	王永

编号	版本	内容	日期	修改人
9	0.71	writeUserDefinedData 接口可输入的字节数从 4 扩展为 6； 补充 BaseActivity 类的 API 说明； 修改按键附表，改为表格形式	2014-1-13	王永
10	0.72	libremote_client_sdk.so 改名为：libexdeviceservicesdk.so 补充对模拟系统事件 API 的说明	2014-1-16	王永
11	0.80	根据外设标准 1.2.3 做如下修改： 1、EXDeviceManager 增加“期望设备属性”的设置 API：setExpectedDeviceProperty； 2、EXDevice 增加对外设的回写：设置握向 setOrientationState，设置开关系统鼠标 setEnableSysMouse 3、EXDevice 增加获取外设类型的 API：getDevicetType 4、AMouseEvent.AMouseData 增加中键及其对应的值 5、增加手柄外设的示意图	2014-1-20	王永
12	0.81	EXDeviceManager 类增加 isHostDeviceModeEnabled 方法，判断当前是否开启了主设备模式	2014-1-23	王永
13	0.82	EXDevice 类增加判断外设是否具备某些 features 的接口： boolean hasFeatures(long features)	2014-2-13	王永
14	0.83	1、EXDeviceManager 类增加获取当前初始化状态的接口： int getInitState() 2、BaseActivity 类去掉获取 SDK 是否初始化成功的接口 boolean isSDKInit()	2014-2-14	王永
15	0.84	1、增加对监听回调的说明：与调用者在同一线程； 2、增加页眉和水印	2014-2-27	王永
16	1.01	增加一些设备类型定义	2014-3-27	王永

编号	版本	内容	日期	修改人
17	1.02	EXDevice 类增加了发送多点 Touch 系统事件的接口等	2014-3-31	王永
18	1.04	Global 类增加 TV 助手 Type 定义 完善 setVibrate 接口定义	2014-6-7	王永
19	1.05	整合加入支付 sdk 文档	2014-11-20	许科&熊志勇
20	1.06	增加阿里手柄键值映射	2014-12-11	许科
21	1.07	加入使用注意，需开发者对 Activity 的手柄键值做返回 true 处理	2015-01-09	许科
22	1.08	加入使用注意，如开发者需要对 yunos 系统做单独处理，可参照环境配置里的注意 2	2015-01-21	许科
23	1.09	增加 CP 接入外设 SDK 和外设兼容的验收标准和注意事项	2015-2-15	许科
24	1.10	修改 yunos 系统判断条件	2015-3-2	许科

目录

第一部分 外设 sdk

1. 简介

《阿里 TV 外设 SDK 使用手册》（以下简称手册或本手册）是阿里 TV OS 外设 SDK API 的介绍文档。开发者可基于本手册，在阿里 TV OS 上开发或适配出支持阿里 TV 外设操控的 APP。外设开发商请参考《阿里 TV 体感手柄外设标准》（以下简称外设标准）。

本文档对主要的 API 做了介绍，未介绍的 API 一般可以自行理解或不需要理解。

本 sdk 主要为了更好的适配体感（陀螺仪）、多人（双人游戏）、震动和语音功能的游戏和应用。

如果 APP 没有如上需求，不使用 SDK 接入方式，至少需要支持**《3 阿里 TV 标准手柄/遥控器键值》**的手柄键值以使用阿里 TV 外设

2. SDK 环境配置

本手册仅适用于安装有阿里 TV 外设驱动的阿里 TV 设备，且仅适用于阿里 TV OS 的 Java 开发环境。环境配置步骤如下：

1、将 exdeviceservicesdk.jar 放入工程的 libs 目录下，右键点击该 jar 包，选择 Build Path -> Add to Build Path；

2、将 libexdeviceservicesdk.so 放入工程的 libs/armeabi/目录下；

3、在工程的 AndroidManifest 文件中添加如下 permission：

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

环境配置结束，接下来就可以使用 exdeviceservicesdk.jar 提供的 API 进行开发。

注意：

1.使用 API 接口时，需要游戏屏蔽 Activity 框架的手柄相关键值，即对 OnKeyDown 或 dispatchKeyEvent 中 KEYCODE_BUTTON_A~KEYCODE_BUTTON_MODE 及 KEYCODE_BUTTON_1~KEYCODE_BUTTON_16 的键值做返回 true 的处理.

2.如果第三方游戏开发者想判断是否是 yunos 系统，可以用如下接口

```
static boolean isYunOSSystem() {
    try {
      Class<?> sp = Class.forName("android.os.SystemProperties");
      Method m = sp.getDeclaredMethod("get", String.class);
      if (m.invoke(sp, "ro.yunos.product.chip") != null || m. invoke(sp, "ro.yunos.hardware") != null)
        return true;
    } catch (Exception e) { }

    return false;
  }
```

3. 阿里 TV 标准手柄/遥控器键值

键值适配是应用和游戏适配阿里 TV 终端的基本技术标准；

适配要求：TV 应用和 TV 游戏至少需要适配遥控器和手柄两个外设中的一个

Linux 和 Android 键值是键值适配的两个标准，建议优先适配 Linux 键值，同时兼容 Android

手柄键值

Linux 键值(scan code)	Android 键值	备注
KEY_LEFT (105)	KEYCODE_DPAD_LEFT	方向左键
KEY_UP (103)	KEYCODE_DPAD_UP	方向上键
KEY_RIGHT (106)	KEYCODE_DPAD_RIGHT	方向右键
KEY_DOWN (108)	KEYCODE_DPAD_DOWN	方向下键
KEY_MENU (139)	KEYCODE_MENU	菜单键
KEY_HOMEPAGE (172)	KEYCODE_HOME	主页键
KEY_ESC (001)	KEYCODE_BACK	返回键
KEY_ENTER (028)	KEYCODE_DPAD_CENTER	确认键
BTN_A (0x130)	KEYCODE_BUTTON_A	游戏 A Button 键
BTN_B (0x131)	KEYCODE_BUTTON_B	游戏 B Button 键
BTN_X (0x133)	KEYCODE_BUTTON_X	游戏 X Button 键
BTN_Y (0x134)	KEYCODE_BUTTON_Y	游戏 Y Button 键
BTN_TL (0x136)	KEYCODE_BUTTON_L1	游戏 L1 Button 键
BTN_TL2 (0x138)	KEYCODE_BUTTON_L2	游戏 L2 Button 键
BTN_TR (0x137)	KEYCODE_BUTTON_R1	游戏 R1 Button 键
BTN_TR2 (0x139)	KEYCODE_BUTTON_R2	游戏 R2 Button 键
BTN_SELECT (0x13a)	KEYCODE_BUTTON_SELECT	游戏 Select Button 键
BTN_START (0x13b)	KEYCODE_BUTTON_START	游戏 Start Button 键
BTN_THUMBL (0x13d)	KEYCODE_BUTTON_THUMBL	Left Thumb Button
BTN_THUMBR (0x13e)	KEYCODE_BUTTON_THUMBR	Right Thumb Button

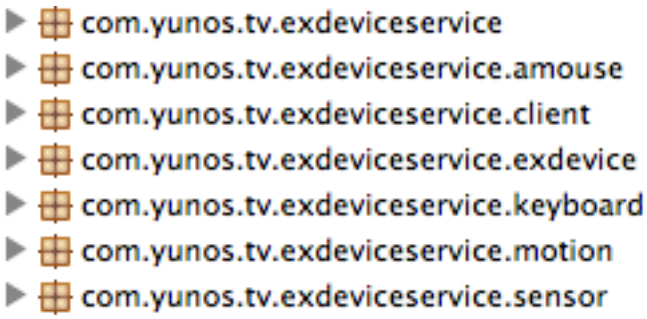
手柄轴号

Linux 轴号	Android 轴号	备注
ABS_X (0x00)	AXIS_X	左摇杆左右
ABS_Y (0x01)	AXIS_Y	左摇杆上下
ABS_Z (0x02)	AXIS_Z	右摇杆左右
ABS_RZ (0x05)	AXIS_RZ	右摇杆上下
ABS_BREAK (0x0a)	AXIS_BREAK	左扳机键
ABS_GAS (0x09)	AXIS_GAS	右扳机键
ABS_HAT0X (0x10)	AXIS_HAT_X	十字键左右
ABS_HAT0Y (0x11)	AXIS_HAT_Y	十字键上下

4. SDK API 介绍

1. Jar 包介绍

Jar 文件结构如下：



主要 API 集中在".client"子包的 EXDeviceManager 和 EXDevice 类中，后面会详细介绍其 API，这个子包中还包含了 OnAMouseListener 等 5 个监听事件接口。其他诸如".amouse"、".exdevice"、".keyboard"、".motion"、".sensor"子包分别定义了上述的 5 种事件。

2. IEvent 类

IEvent 为其他监听事件的父类，开发者不需要直接操作该类。需要关心其定义的两个 API，这两个 API 同样可以被它的子类调用：

获取外设 ID：

public int getDeviceId()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的 ID	可用于操作外设的 API，表示外设序号等

获取 Raw data：

public RawData getRawData()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
RawData		事件上传的 Raw Data，详见 RawData 类介绍	定义于 com.yunos.tv.exdeviceservice.exdevice

3. Global 类

com.yunos.tv.exdeviceservice.Global 类中开发者需要关注以下内容：

1、外设类型

```
public static final int DEV_SINGLE_HAND_HANDLE = 0; // 单手手柄
public static final int DEV_PAIR_HAND_HANDLE = 1; // 双手手柄
public static final int DEV_DANCE_MAT = 2; // 跳舞毯
public static final int DEV_YOGA_BLANKET = 3; // 瑜伽毯
public static final int DEV_STEERING_WHEEL = 4; // 方向盘
public static final int DEV_TOY_GUN = 5; // 玩具枪
public static final int DEV_SOMATOSENSORY_STICK = 6; // 体感棒
public static final int DEV_ANALOG_GUITAR = 7; // 模拟吉他

public static final int DEV_TV_HELPER = 32; // TV 助手
```

2、EXDevice 定义的各 feature 值：

一个 EXDevice 由多个 feature 构成，通过 EXDevice 对象可以查询到支持的 features，feature 值的定义如下：

```
public static final long DEV_FEA_KEYBOARD = 0x0001;     //键盘
public static final long DEV_FEA_AMOUSE = 0x0002;       //鼠标
public static final long DEV_FEA_TOUCHPAD = 0x0004;     //触控板
public static final long DEV_FEA_MOTIONSTICK = 0x0008; //摇杆
public static final long DEV_FEA_MSENSOR = 0x0010;      //传感器
public static final long DEV_FEA_VIBRATOR = 0x0020;     //震动马达
public static final long DEV_FEA_MIC = 0x0040;          //麦克风
```

各 feature 按位与构成了一个 EXDevice 的 features 值。

3、震动马达的类型：

```
public static final int VIBRATOR_LEFT = 0x01;          //左马达
public static final int VIBRATOR_RIGHT = 0x02;         //右马达
```

与 feature 一样，马达类型也可以按位与，VIBRATOR_LEFT | VIBRATOR_RIGHT 表示同时左右两个马达，在 EXDevice 的操作马达 API 中，可以通过这种方式传入需要操作的马达的类型。

4、麦克风的操作 Flag 值：

```
public static final int MIC_OPEN = 0;       //开启麦克风
public static final int MIC_CLOSE = 1;      //关闭麦克风
```

在 EXDevice 的操作麦克风的 API 中使用。

5、Log 开关

```
public static boolean ENABLE_LOGD = false;
public static boolean ENABLE_LOGI = false;
public static boolean ENABLE_LOGE = false;
```

SDK 中的部分 Log 的开关，默认为 false，关闭状态，Log 的 TAG 为“EXDeviceServiceSDK”，开发者调试阶段可以打开，查看 SDK 的部分 Log 输出，正式发布请务必关闭 Log，对性能影响较大。

4. RawData 类

成员名	类型	语义	备注
mDeviceId	int	外设的 ID	
mType	int	数据类型	0: Joystick 1: Sensor 2: Key 3: Mouse 4. EXDevice
mData	byte[]	字节数据	

属于特殊数据，普通开发者不需要使用，特殊开发者请与我们联系。该类描述了随事件传送的 Raw 数据，当前表示相对于该事件驱动层收到的字节数据。其成员变量如下：

5. 监听事件类

SDK 支持对 EXDevice 的各 feature 的事件注册监听器，通过监听事件及时得到 EXDevice 的数据。

5.1 传感器监听（OnMSensorListener）

传感器事件类（MSensorEvent）：

该类定义了驱动层发送的六轴或九轴传感器（加速度、重力、磁力传感器，需要外设支持）事件。主要描述了事件的数据，使用了内部类 MSensorData，获取数据的 API 如下：

获取事件对应的数据：

`public MSensorData getEventData()`			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
MSensorData		传感器事件的数据	具体各轴的数据调用 MSensorData 类提供的 getValue(int type)接口

内部类 MSensorData 的主要 API 如下：

获取指定轴类型的 Sensor 数据：

`public float getValue(int type)`			
输入参数	**类型**	**语义**	**备注**

public float getValue(int type)			
type	int	指定的轴类型	type 定义于 MSensorEvent 类，值如下： public static final int ACCE_AXIS_X = 0; public static final int ACCE_AXIS_Y = 1; public static final int ACCE_AXIS_Z = 2; public static final int GYRO_AXIS_X = 3; public static final int GYRO_AXIS_Y = 4; public static final int GYRO_AXIS_Z = 5; public static final int MAGN_AXIS_X = 6; public static final int MAGN_AXIS_Y = 7; public static final int MAGN_AXIS_Z = 8;
返回值类型			
float		对应轴的数据	不支持的轴或无数据，返回统一的无效值： float MSENSOR_INVALID_DATA = 3.4E38F ACCE 加速度传感器返回的是以 g 为单位的加速度值，GYRO 陀螺仪传感器返回的是角速度值

判断轴数据是否有效：

public static boolean isValidValue(float v)			
输入参数	**类型**	**语义**	**备注**
v	float	轴数据	

public static boolean isValidValue(float v)		
返回值类型		
boolean	true: 有效 false：无效	

传感器事件监听器（OnMSensorListener）：

```
public interface OnMSensorListener {
      public void onMSensor(MSensorEvent event);
}
```

5.2 Joystick 监听（OnJMotionListener）

Joystick 事件类（JMotionEvent）：

该类定义了驱动层发送的 Joystick 事件，主要描述了事件的数据，使用了内部类 JMoitonData，最大可支持 8 轴数据，获取数据的 API 如下：

获取事件对应的数据：

public JMotionData getEventData()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
JMotionData		Joystick 事件的数据	具体各轴的数据调用 JMotionData 类提供的 getValue(int type)接口

内部类 JMotionData 的主要 API 如下：

获取指定轴类型的数据：

public float getValue(int type)			
输入参数	**类型**	**语义**	**备注**

public float getValue(int type)			
type	int	指定的轴类型	type 定义于 JMotionEvent 类，值如下： public static final int AXIS_LX = 0; public static final int AXIS_LY = 1; public static final int AXIS_RX = 2; public static final int AXIS_RY = 3; public static final int AXIS_LT = 4; public static final int AXIS_RT = 5; public static final int AXIS_HAT_X = 6; public static final int AXIS_HAT_Y = 7;
返回值类型			
float		对应轴的数据	不支持的轴或无数据，返回统一的无效值： float JMOTION_INVALID_DATA = 3.4E38F 轴值范围：[-1.0f, 1.0f]

判断轴数据是否有效：

public static boolean isValidValue(float v)			
输入参数	**类型**	**语义**	**备注**
v	float	轴数据	
返回值类型			
boolean		true: 有效 false：无效	

Joystick 事件监听器（OnJMotionListener）：

```
public interface OnJMotionListener {
      public void onJMotion(JMotionEvent event);
}
```

5.3 按键监听（OnDKeyListener）

普通按键事件类（DKeyEvent）：

该类定义了驱动层发送的按键状态变化事件，主要描述了事件的数据，使用了内部 DKeyData，获取数据的 API 如下：

获取事件对应的数据：

public DKeyData getEventData()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			

DKeyData		按键事件的数据	DKeyData 公有成员如下：
			public int mSize; //数组的大小
			public int[] mKeyCodes; //键值数组
			public int[] mActions; //键 Action 状态数组
			键值定义见文档后面的附表
			键 Action 状态定义在 DKeyEvent 类，如下：
			public static final int ACTION_UNKNOWN = -1;
			public static final int ACTION_DOWN = 0;
			public static final int ACTION_UP = 1;

组合键（CombKey）：

支持将两个或以上的按键定义为一个组合键，并将检测按键事件是否符合已定义的组合键，符合则发送组合键事件。组合键提供了两个构造方法：

public CombKey(String id, int key1, int key2)			
输入参数	**类型**	**语义**	**备注**
id	String	组合键 ID，作为标识，组合键之间不能重复	不能为 null 或 length 为 0
key1	int	第一个键的键值	键值参考本文档后面的附表

public CombKey(String id, int key1, int key2)			
key2	int	第二个键的键值	键值参考本文档后面的附表
返回值类型			

public CombKey(String id, int key1, int key2, int key3)			
输入参数	**类型**	**语义**	**备注**
id	String	组合键 ID，作为标识	组合键之间不能重复
key1	int	第一个键的键值	键值参考本文档后面的附表
key2	int	第二个键的键值	键值参考本文档后面的附表
key3	int	第三个键的键值	键值参考本文档后面的附表
返回值类型			

如果需要构造多于三个按键的组合键，可以使用以下方法：

public void addKey(int newKey)			
输入参数	**类型**	**语义**	**备注**
newKey	int	向当前组合键对象添加新的键值	不能与当前已存在的键值相同
返回值类型			
void			

组合键提供查询 API，主要如下：

public String getId()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
String		组合键的 id	

组合键事件（CombKeyEvent）：

该类定义了组合键事件，事件的数据为组合键对象，获取组合键对象的 API 如下：

public CombKey getCombKey()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
CombKey		事件对应的组合键	

按键事件监听器（OnDKeyListener）：

```
public interface OnDKeyListener {
      public void onDKey(DKeyEvent event);
      public void onCombKey(CombKeyEvent event);
}
```

5.4 鼠标监听（OnAMouseListener）

鼠标事件类（AMouseEvent）：

该类定义了驱动层发送的鼠标事件，主要描述了事件的数据，使用了内部类 AMouseData，获取数据的 API 如下：

获取事件对应的数据：

public AMouseData getEventData()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
AMouseData		鼠标事件的数据	AMouseData 公有成员如下： public int mPosX;　//X 绝对坐标值 public int mPosY;　//Y 绝对坐标值 public int mMidValue;　　//中键滚轮的值 public int[] mKeyStates; //左、右、中键状态

鼠标键值及状态定义在 AMouseEvent 类，如下：

```
    //左右中键
    public static final int MOUSE_LEFT_KEY = 0;
    public static final int MOUSE_RIGHT_KEY = 1;
    public static final int MOUSE_MID_KEY = 2;
    //左右中键状态值
    public static final int MOUSE_RELEASED = 0;
    public static final int MOUSE_PRESSED = 1;
```

内部类 AMouseData 的主要 API 如下：

判断轴数据是否有效：

```
public static boolean isValidValue(float v)
```

输入参数	类型	语义	备注
v	float	轴数据	
返回值类型			
boolean		true: 有效 false：无效	适用于 AMouseData 各成员值的判断

鼠标事件监听器（OnAMouseListener）：

```
public interface OnAMouseListener {
    public void onAMouse(AMouseEvent event);
}
```

5.5 外设状态监听（OnEXDeviceListener）

外设状态事件（EXDeviceEvent）

该类定义了驱动层发送的外设状态事件，主要描述了事件的数据，使用了内部类 EXDeviceData，获取数据的 API 如下：

获取事件对应的数据：

```
public EXDeviceData getEventData()
```

输入参数	类型	语义	备注

public EXDeviceData getEventData()			
无			
返回值类型			

EXDeviceData		事件的数据	EXDeviceData 公有成员如下：
			public int mConnectionState; //外设连接状态
			public int mPowerPercent; //外设电量百分比

外设连接状态值定义如下：

```
public static final int DEV_STATE_UNKONWN = -1;
public static final int DEV_STATE_DISCONNECTED = 0;
public static final int DEV_STATE_CONNECTING = 1;
public static final int DEV_STATE_CONNECTED = 2;
```

外设状态事件监听器（OnEXDeviceListener）：

```
public interface OnEXDeviceListener {
    public void onEXDevice(EXDeviceEvent event);
}
```

6. EXDeviceManager 类

该类是 SDK 中主要 API 提供方，提供了诸如事件监听器的注册，外设列表的获取，初始化，反初始化等 API，是开发者首先使用的类。监听器受限于外设对相应 feature 的支持，对不支持的 feature 注册监听器将收不到相应的监听事件。SDK 对注册进来的监听器的回调和开发者调用 getInstance 方法在同一个线程中。

该类采用单例模式实现，获取单例的 API 为：

public static EXDeviceManager getInstance()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
EXDeviceManager		返回 EXDeviceManager 实例	通过该实例操作其他 API

需要开发者注意的是，在你的组件的 onResume 和 onPause 方法中，应该对 EXDeviceManager 的单例进行初始化和反初始化，否则，可能出现不可预见的错误，分别调用如下 API：

public void init(Context appContext, InitListener listener)			
输入参数	**类型**	**语义**	**备注**
appContext	Context	环境上下文	传入不可为 null，否则抛 IllegalArgumentException
listener	InitListener	初始化监听器	不可为 null，否则抛 IllegalArgumentException
返回值类型			
void			

InitListener 初始化状态监听：

```
public static interface InitListener {
            //初始化成功回调
            public void onSuccess();
            /*
            初始化失败回调，初始化失败的可能原因：
            1. 没有运行的阿里 TV OS 环境下或没有安装外设驱动
            2. 外设驱动发生错误停止运行
            3. Manifest 没有添加 INTERNET permission
            */
            public void onFailed();
      }
```

public void deinit()			
输入参数	**类型**	**语义**	**备注**
无			init 对应的方法，请务必调用
返回值类型			
void			

获取 SDK 初始化状态：

public int getInitState()			
输入参数	**类型**	**语义**	**备注**

public int getInitState()			
无			
返回值类型			
int		SDK 初始化状态	状态值定义在 EXDeviceManager 类中： public static final int STATE_NOT_INIT = 0; public static final int STATE_INITING = 1; public static final int STATE_INITED = 2;

设置期望的外设属性：

public void setExpectedDeviceProperty(boolean isHorizontal, long features, boolean enableSysMouse)			
输入参数	**类型**	**语义**	**备注**
isHorizontal	boolean	是否期望使用外设的横向模式，能否满足期望视外设而定	默认值为 false，即竖直模式。对于诸如赛车之类的游戏，横向使用游戏手柄更符合体验，应该设置该值为 true。 单手手柄：具有横向和竖直两种模式； 双手手柄：只具有横向模式 注意：如果你设置了横向模式，请务必在 deinit 之前将其重置为竖向模式

public void setExpectedDeviceProperty(boolean isHorizontal, long features, boolean enableSysMouse)			
features	long	期望外设具有的 features，使用“位或”传入，请务必配置好需要使用到的 feature，否则将不能通过监听或查询得到。	外设 feature 定义见上面的 Global 类介绍，默认值为： Global.DEV_FEA_KEYBOARD |Global.DEV_FEA_MOTIONSTICK | Global.DEV_FEA_MSENSOR；如果需要监听鼠标事件，请’或’上 Global.DEV_FEA_AMOUSE，否则将接收不到该事件
enableSysMouse	boolean	期望关闭外设的系统鼠标事件	外设标准规定，空鼠事件同时发送给系统，对于某些游戏中，可能并不希望看到系统的鼠标图标，可以设置为 false； 默认值为 false
返回值类型			

获取外设的列表：

public List<EXDevice> getDeviceList()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
List<EXDevice>		外设对象列表	通过外设 EXDevice 对象可以对外设进行具体操作，详见 EXDevice 类的介绍

获取外设的数目：

public int getDeviceCount()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的数目	该方法等同于 getDeviceList() 返回的列表 size

通过 DeviceId 获取 EXDevice 对象：

public EXDevice getDevice(int deviceId)			
输入参数	**类型**	**语义**	**备注**
deviceId	int	外设的 ID	
返回值类型			
EXDevice		外设的对象	

注册传感器监听器：

public void setOnMSensorListener(OnMSensorListener listener)			
输入参数	**类型**	**语义**	**备注**
listener	OnMSensorListener	传感器事件监听器	参阅文档前面部分
返回值类型			
void			

注册 Joystick 监听器：

public void setOnJMotionListener(OnJMotionListener listener)			
输入参数	**类型**	**语义**	**备注**
listener	OnJMotionListener	Joystick 事件监听器	参阅文档前面部分
返回值类型			
void			

注册按键监听器：

```
public void setOnDKeyListener(OnDKeyListener listener)
```

输入参数	类型	语义	备注
listener	OnDKeyListener	按键事件监听器	参阅文档前面部分
返回值类型			
void			

注册鼠标监听器：

```
public void setOnAMouseListener(OnAMouseListener listener)
```

输入参数	类型	语义	备注
listener	OnAMouseListener	按键事件监听器	参阅文档前面部分
返回值类型			
void			

注册外设状态监听器：

```
public void setOnEXDeviceListener(OnEXDeviceListener listener)
```

输入参数	类型	语义	备注
listener	OnEXDeviceListener	外设状态监听器	参阅文档前面部分
返回值类型			
void			

SDK 支持主设备模式，在该模式下 SDK 只接收主设备的事件。默认该模式为关闭状态。未主动设置主设备的情况下，默认为设备列表的第一个设备为主设备。相关 API 如下：

主设备模式使能接口：

```
public boolean setEnableHostDeviceMode(boolean enable)
```

public boolean setEnableHostDeviceMode(boolean enable)			
输入参数	**类型**	**语义**	**备注**
enable	boolean	开启或关闭主设备模式	
返回值类型			
boolean		是否成功执行	

获取是否开启了主设备模式

public boolean isHostDeviceModeEnabled()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
boolean		是否开启了主设备模式	

设置主设备：

public boolean setHostDeviceId(int deviceId)			
输入参数	**类型**	**语义**	**备注**
deviceId	int	待设置为主设备的设备 ID	
返回值类型			
boolean		是否设置成功	默认设置主设备为设备列表的第一个设备

获取主设备的 EXDevice 对象：

public EXDevice getHostDevice()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
EXDevice		主设备对象	

获取主设备的设备 ID：

public int getHostDevieId()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		主设备的设备 ID	

设置系统事件模式：

开启系统事件模式：外设的事件走系统标准事件，这些事件包括：按键、鼠标

关闭系统事件模式：外设的所有事件通过 SDK 的 Listener 发出，不走系统标准事件

默认该模式是关闭的，建议只在游戏的菜单界面开启，退出界面后请关闭。

public boolean setEnableSysEventMode(boolean enable)			
输入参数	**类型**	**语义**	**备注**
enable	boolean	是否开启系统事件模式	传感器事件始终走 SDK 的 Listener，与该模式的开启或关闭无关
返回值类型			
boolean		是否执行成功	

组合键的注册：

public boolean registerCombKey(CombKey ck)			
输入参数	**类型**	**语义**	**备注**
ck	CombKey	组合键对象	参阅文档前面部分
返回值类型			
boolean		是否注册成功	

组合键的注销：

public boolean unregisterCombKey(String combKeyId)			
输入参数	**类型**	**语义**	**备注**
combKeyId	String	组合键对象的 ID	参阅文档前面部分
返回值类型			
boolean		是否注销成功	

获取 SDK 的版本号 Code：

public int getVersionCode()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		版本号代码	

获取 SDK 的版本号 Name：

public String getVersionName()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
String		版本号名称	

7. EXDevice 类

该类主要定义了对外设操作的 API 接口，包括查询外设的 features、连接状态、电量、按键的 Action 状态、传感器数据、Joystick 数据、鼠标数据、控制震动器、开关 feature、获取外设的厂商及外设本身的信息等。

获取设备 ID

public int getDeviceId()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的 ID	外设的 ID 指外设在阿里 TV 上的序号

获取设备类型

public int getDevicetType()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的类型	参见前面 Global 类的介绍

获取外设的厂商名称：

public String getVendorName()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
String		外设的厂商名称	

获取外设的厂商 ID:

public int getVendorId()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			

public int getVendorId()		
int	外设的厂商 ID	

获取外设的产品名称：

public String getProductName()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
String		外设的产品名称	

获取外设的产品 ID:

public int getProductId()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的产品 ID	

获取外设支持的 Feature：

public long getDeviceFeatures()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
long		外设的功能属性，返回值是各功能 feature 的“或”结果	feature 定义参见文档前面的 Global 类定义

判断外设是否支持某些 Features：

public boolean hasFeatures(long features)

public boolean hasFeatures(long features)			
输入参数	**类型**	**语义**	**备注**
features	long	feature 的“或”值	可以同时判断是否支持多个 features，将不同 feature 值相”或“，如： hasFeatures(Global.DEV_FEA_MSENSOR | Global.DEV_FEA_VIBRATOR) 即判断外设是否都支持 sensor 和振动
返回值类型			
boolean		是否支持传入的 features	

获取外设的连接状态：

public int getConnState()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的连接状态值	连接状态值参见文档前面的 EXDeviceEvent 类的介绍

获取外设的电量百分比：

public int getPowerPercent()			
输入参数	**类型**	**语义**	**备注**
无			
返回值类型			
int		外设的电量百分比	

获取按键的 Action 状态：

```
public int getKeyAction(int keyCode)
```

输入参数	类型	语义	备注
keyCode	int	键值	
返回值类型			
int		按键 Action 状态	取值参见文档前面的 DKeyEvent 介绍

获取鼠标的 X 坐标位置：

```
public int getAMousePosX()
```

输入参数	类型	语义	备注
无			
返回值类型			
int		X 坐标位置	屏幕的绝对坐标

获取鼠标的 Y 坐标位置：

```
public int getAMousePosY()
```

输入参数	类型	语义	备注
无			
返回值类型			
int		Y 坐标位置	屏幕的绝对坐标

获取鼠标的中键滚动值：

```
public int getAMouseMidValue()
```

输入参数	类型	语义	备注
无			

public int getAMouseMidValue()		
返回值类型		
int	中键滚动值	

重置鼠标的起始坐标点

public boolean resetAMousePos(int x, int y)			
输入参数	**类型**	**语义**	**备注**
x	int	X 坐标值	
y	int	Y 坐标值	
返回值类型			
boolean		方法是否执行成功	

获取鼠标的按键状态

public int getAMouseState(int keyFlag)			
输入参数	**类型**	**语义**	**备注**
keyFlag	int	左右键 Flag	参见 AMouseEvent public static final int MOUSE_LEFT_KEY = 0; public static final int MOUSE_RIGHT_KEY = 1;
返回值类型			
int		对应 keyFlag 的状态值	参见 AMouseEvent public static final int MOUSE_RELEASED = 0; public static final int MOUSE_PRESSED = 1; public static final int MOUSE_INVALID_DATA = 0x7FFFFFFF

获取 Joystick 指定轴的数据

public float getJMotionValue(int valueType)			
输入参数	**类型**	**语义**	**备注**
valueType	int	Joystick 轴类型	参见 JMotionEvent
返回值类型			
float		轴数据	外设不支持 Joystick feature 或不支持的轴类型，返回无效值，参见 JMotionEvent 介绍

获取 Joystick 的所有轴的数据

public int getJMotionValues(float[] values)			
输入参数	**类型**	**语义**	**备注**
values	float[]	返回 Joystick 的 8 轴数据	得到的值可能为无效，视设备支持的能力而定，请使用 JMotionData.isValidValue 判定值的有效性
返回值类型			
int		实际写入的 values 的 length	SDK 支持 8 轴的 Joystick 数据，如传入的 values 的 length 为 6，则返回 6

获取 Sensor 指定轴的数据

public float getMSensorValue(int valueType)			
输入参数	**类型**	**语义**	**备注**
valueType	int	sensor 轴类型	参见 MSensorEvent
返回值类型			
float		轴数据	外设不支持 MSensor feature 或不支持的轴类型，返回无效值，参见 MSensorEvent 介绍

获取 Sensor 的所有轴数据

public float getMSensorValues(float[] values)			
输入参数	**类型**	**语义**	**备注**

public float getMSensorValues(float[] values)			
values	float[]	返回 Sensor 的 9 轴数据	得到的值可能为无效，视设备支持的能力而定，请使用 MSensorData.isValidValue 判定值的有效性
返回值类型			
int		实际写入的 values 的 length	SDK 支持 9 轴的 Sensor 数据，如传入的 values 的 length 为 6，则返回 6

外设 Feature 的使能接口

public boolean setEnableFeatures(long features, boolean enable)			
输入参数	**类型**	**语义**	**备注**
features	long	外设 feature 的“或”值	参见 Global 类关于 feature 的定义
enable	boolean	开启或关闭	
返回值类型			
boolean		执行成功与否。	目前支持的使能接口的 feature 如下： DEV_FEA_AMOUSE, DEV_FEA_MSENSOR, DEV_FEA_MIC 不同设备对该 API 的支持特性可能不同，建议将该方法只作为辅助用户体验使用。

震动马达操作接口

public boolean setVibrate(int vibratorTypes, int mode, long milliSec)			
输入参数	**类型**	**语义**	**备注**

public boolean setVibrate(int vibratorTypes, int mode, long milliSec)			
vibratorType	int	马达的类型	参见 Global 类关于震动马达类型的介绍。 注意： SDK 根据 type 相应的“位值”来确定是开启还是关闭对应的马达。 如： 传入 0，所有马达关闭； 传入 1, 表示仅开启 1 号马达，其余关闭； 传入 2，表示仅开启 2 号马达，其余关闭； 传入 3， 表示仅开启 1，2 号马达，其余关闭； 依次类推
mode	int	震动模式	0 - 255 表示震动强度 0：表示默认震动强度 1 - 255 表示震动强度从弱到强，1 为最弱，255 为最强
milliSec	long	震动时长	单位为毫秒（视设备实现情况，震动时长有可能不精准，一般建议震动时长 100ms - 10000ms 之间） -1：表示持续震动
返回值类型			
boolean		执行成功与否	如不支持震动马达 feature，则直接返回 false

回写外设自定义格式的数据:

属于特殊通道，外设厂商需要与我们联系开通，才能使用。

public boolean writeUserDefinedData(byte[] data)			
输入参数	**类型**	**语义**	**备注**

public boolean writeUserDefinedData(byte[] data)			
data	byte[]	待回写的 byte 数组	最大不能超过 6 个字节，数据格式由外设厂商定义，SDK 只负责写入操作，请参考相应的外设 Spec 文档
返回值类型			
boolean		执行成功与否	需要设备支持，非通用接口

设置外设的握向

public boolean setOrientationState(boolean isHorizontal)			
输入参数	**类型**	**语义**	**备注**
isHorizontal	boolean	是否设置为横向	true：横向；false：竖向
返回值类型			
boolean		执行成功与否	是否能设置成功，视具体外设支持情况而定； 单手手柄支持横向和竖向；双手手柄只支持横向 注意：当你设置某个设备为横向模式时，请务必在 deinit 之前将其设置回竖向模式

设置是否启用外设的系统鼠标

public boolean setEnableSysMouse(boolean enable)			
输入参数	**类型**	**语义**	**备注**
enable	boolean	启用与否	true：启用；false：关闭
返回值类型			
boolean		执行成功与否	支持空鼠的标准外设，会同时给系统和 SDK 发送鼠标事件，本 API 为控制是否给系统发送鼠标事件

发送系统的按键事件

属于特殊 API，开发者需要与我们联系开通，才能使用。

public boolean sendSysKeyEvent(DKeyEvent event)			
输入参数	**类型**	**语义**	**备注**
event	DKeyEvent	SDK 的按键事件	将 SDK 的按键事件转送为系统的按键事件
返回值类型			
boolean		执行成功与否	

发送系统的 Mouse 事件

属于特殊 API，开发者需要与我们联系开通，才能使用。

public boolean sendSysMouseEvent(AMouseEvent event)			
输入参数	**类型**	**语义**	**备注**
event	AMouseEvent	SDK 的 mouse 事件	将 SDK 的 mouse 事件转送为系统的 mouse 事件
返回值类型			
boolean		执行成功与否	

发送系统的单点 Touch 事件

属于特殊 API，开发者需要与我们联系开通，才能使用。

public boolean sendSysSTouchEvent(STouchEvent event)			
输入参数	**类型**	**语义**	**备注**
event	STouchEvent	SDK 的单点 Touch 事件	将 SDK 的单点 Touch 事件转送为系统的 Touch 事件
返回值类型			
boolean		执行成功与否	

发送系统的多点 Touch 事件

属于特殊 API，开发者需要与我们联系开通，才能使用。

public boolean sendSysMTouchEvent(MTouchEvent event)			
输入参数	**类型**	**语义**	**备注**

public boolean sendSysMTouchEvent(MTouchEvent event)			
event	MTouchEvent	SDK 的多点 Touch 事件	将 SDK 的多点 Touch 事件转送为系统的多点 Touch 事件
返回值类型			
boolean		执行成功与否	

8. BaseActivity 类

该类可作为 app 中使用 Activity 的父类，该类主要帮开发者完成了以下事情：

1、init 和 deinit 的调用，开发者只需要 Override onSuccess 和 onFailed 监听异步的初始化结果；

2、Event 监听器的注册，开发者只需要 Override 对应的 on***Event 方法：onEXDevice, onMSensor, onJMotion, onDKey, onCombKey, onAMouse，即可监听到外设相关事件；

8.1 SDK 初始化

SDK 初始化结果的监听回调

public void onSuccess()			
输入参数	**类型**	**语义**	**备注**
无			初始化成功时回调
返回值类型			

public void onFailed()

public void onFailed()			
输入参数	**类型**	**语义**	**备注**
无			初始化失败时回调
返回值类型			

8.2 事件监听回调

设备状态信息更新事件：

public void onEXDevice(EXDeviceEvent event)			
输入参数	**类型**	**语义**	**备注**
event	EXDeviceEvent		设备事件，可通过该类得到的相关方法和数据
返回值类型			

传感器事件：

public void onMSensor(MSensorEvent event)			
输入参数	**类型**	**语义**	**备注**
event	MSensorEvent		传感器事件，可通过该类得到的相关方法和数据
返回值类型			

Joystick Motion 事件：

public void onJMotion(JMotionEvent event)

public void onJMotion(JMotionEvent event)			
输入参数	**类型**	**语义**	**备注**
event	JMotionEvent		Joystick Motion 事件，可通过该类得到的相关方法和数据
返回值类型			

按键事件：

public void onDKey(DKeyEvent event)			
输入参数	**类型**	**语义**	**备注**
event	DKeyEvent		按键事件，可通过该类得到的相关方法和数据
返回值类型			

组合键事件：

public void onCombKey(CombKeyEvent event)			
输入参数	**类型**	**语义**	**备注**
event	CombKeyEvent		组合键事件，可通过该类得到的相关方法和数据
返回值类型			

鼠标事件：

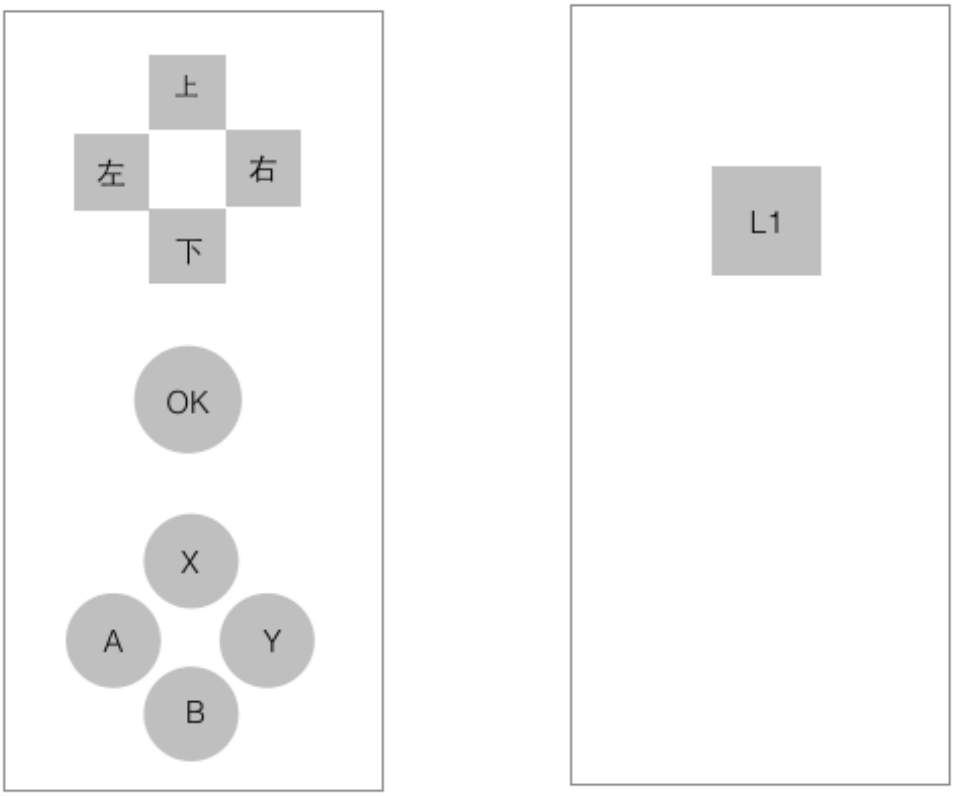
public void onAMouse(AMouseEvent event)			
输入参数	**类型**	**语义**	**备注**
event	AMouseEvent		鼠标事件，可通过该类得到的相关方法和数据
返回值类型			

public void onAMouse(AMouseEvent event)		

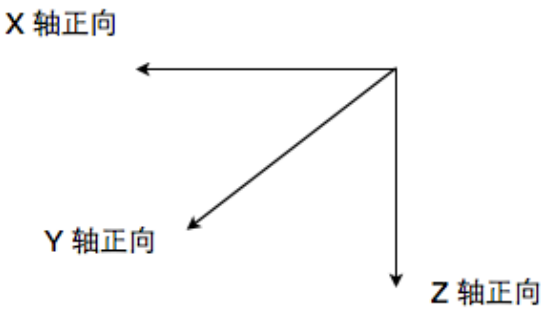
5. 标准外设示意图

1. 单手手柄

外设标准规范了单手手柄在竖向和横向模式下的基本按键和 Sensor 的坐标方向。

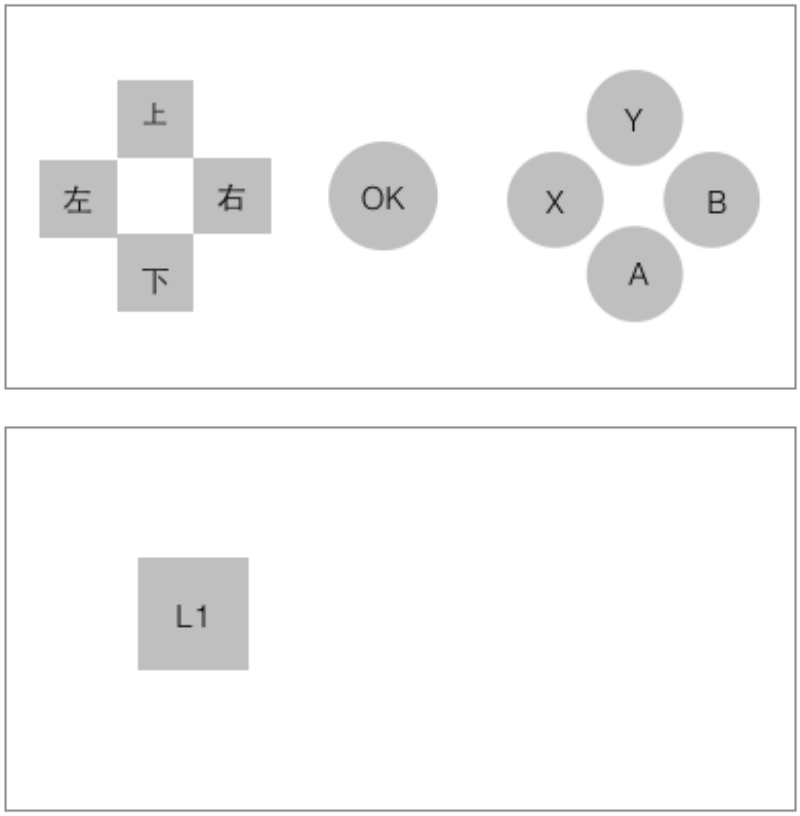


1、竖向模式

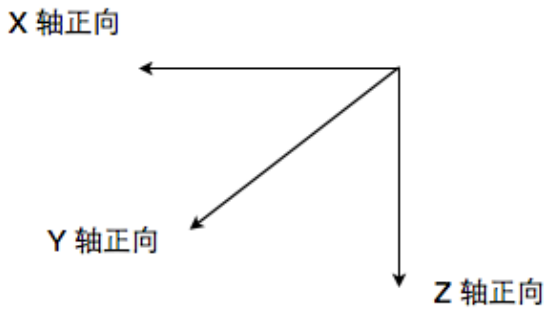


竖向模式下的按键示意图（正面和背面）

竖向模式下的 Sensor 方向示意图



2、横向模式



横向模式下的按键示意图（正面和背面）

横向模式下的 Sensor 方向示意图

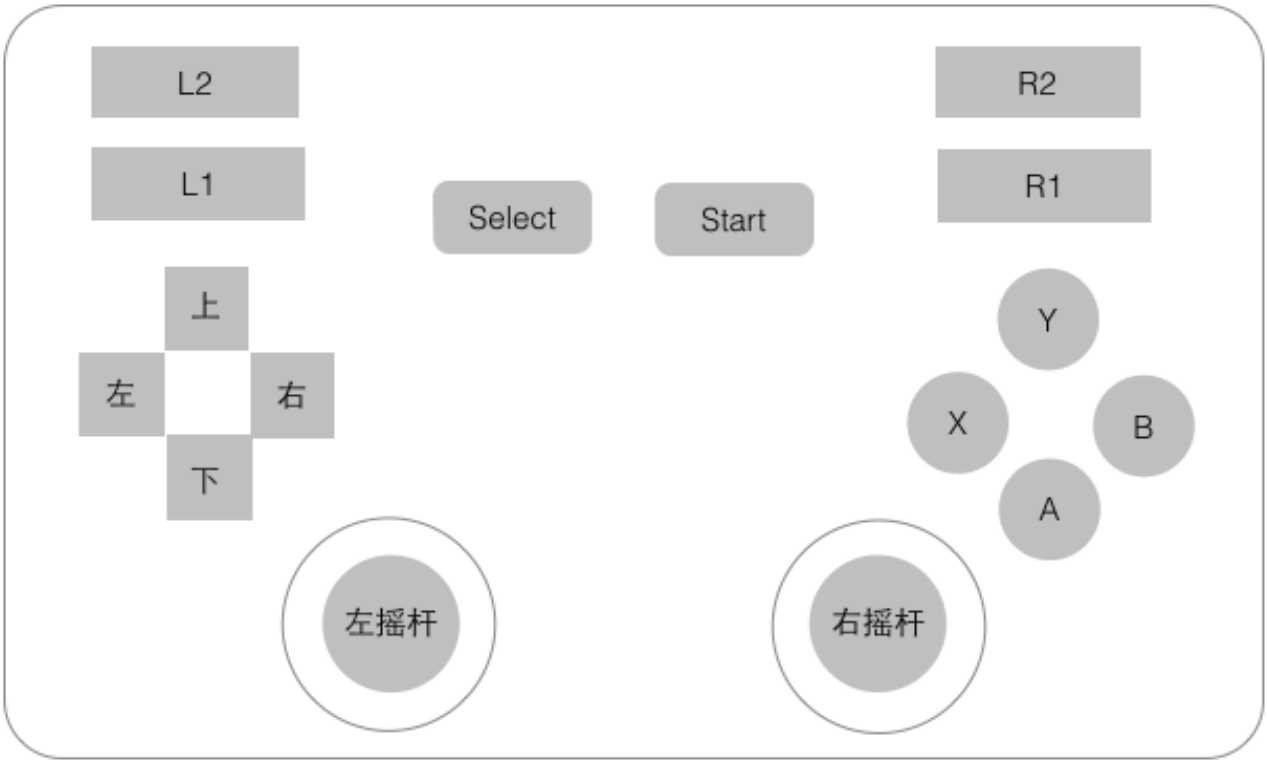
以上根据用户的对手柄的握向，统一了四个方向键和 Sensor 的坐标方向。

两种模式的键值如下(具体对应见下面的键值定义表)：

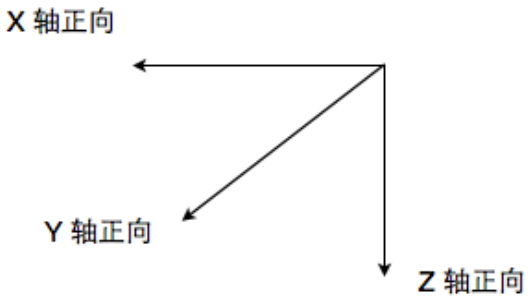
名称	键值	备注
左	KEYCODE_DPAD_LEFT	方向左键

名称	键值	备注
上	KEYCODE_DPAD_UP	方向上键
右	KEYCODE_DPAD_RIGHT	方向右键
下	KEYCODE_DPAD_DOWN	方向下键
OK	KEYCODE_DPAD_CENTER	中键
A	KEYCODE_BUTTON_A	游戏 A Button 键
B	KEYCODE_BUTTON_B	游戏 B Button 键
X	KEYCODE_BUTTON_X	游戏 X Button 键
Y	KEYCODE_BUTTON_Y	游戏 Y Button 键
L1	KEYCODE_BUTTON_L1	游戏 L1 Button 键

2. 双手手柄



外设标准中，双手手柄只有横向模式，基本按键和 Sensor（可选）坐标方向如下：



按键示意图

Sensor 方向示意图

键值表：

名称	键值	备注
左	KEYCODE_DPAD_LEFT	方向左键
上	KEYCODE_DPAD_UP	方向上键
右	KEYCODE_DPAD_RIGHT	方向右键
下	KEYCODE_DPAD_DOWN	方向下键
A	KEYCODE_BUTTON_A	游戏 A Button 键
B	KEYCODE_BUTTON_B	游戏 B Button 键
X	KEYCODE_BUTTON_X	游戏 X Button 键
Y	KEYCODE_BUTTON_Y	游戏 Y Button 键
L1	KEYCODE_BUTTON_L1	游戏 L1 Button 键
L2	KEYCODE_BUTTON_L2	游戏 L2 Button 键
R1	KEYCODE_BUTTON_R1	游戏 R1 Button 键
R2	KEYCODE_BUTTON_R2	游戏 R2 Button 键
Select	KEYCODE_BUTTON_SELECT	游戏 Select Button 键
Start	KEYCODE_BUTTON_START	游戏 Start Button 键
左摇杆按键	KEYCODE_BUTTON_THUMBL	Left Thumb Button
右摇杆按键	KEYCODE_BUTTON_THUMBR	Right Thumb Button

6. 键值定义

SDK 所有支持的键值定义如下（采用了 Android 的定义），通常情况下，外设所支持的键值为它的子集。Key Code 定义在 DKeyEvent 类中，相关定义如下表：

变量名	键值	备注
KEYCODE_UNKNOWN	0	无效或未知按键值
KEYCODE_SOFT_LEFT	1	左软键
KEYCODE_SOFT_RIGHT	2	右软键
KEYCODE_HOME	3	Home 键
KEYCODE_BACK	4	返回键
KEYCODE_CALL	5	呼叫键
KEYCODE_ENDCALL	6	挂机键
KEYCODE_0	7	数字键 0
KEYCODE_1	8	数字键 1
KEYCODE_2	9	数字键 2
KEYCODE_3	10	数字键 3
KEYCODE_4	11	数字键 4
KEYCODE_5	12	数字键 5
KEYCODE_6	13	数字键 6
KEYCODE_7	14	数字键 7
KEYCODE_8	15	数字键 8
KEYCODE_9	16	数字键 9
KEYCODE_STAR	17	* 键
KEYCODE_POUND	18	# 键
KEYCODE_DPAD_UP	19	DPad 上方向键
KEYCODE_DPAD_DOWN	20	DPad 下方向键
KEYCODE_DPAD_LEFT	21	DPad 左方向键
KEYCODE_DPAD_RIGHT	22	DPad 右方向键

变量名	键值	备注
KEYCODE_DPAD_CENTER	23	DPad 中间键
KEYCODE_VOLUME_UP	24	音量增加键
KEYCODE_VOLUME_DOWN	25	音量减小键
KEYCODE_POWER	26	电源键
KEYCODE_CAMERA	27	Camera 键
KEYCODE_CLEAR	28	Clear 键
KEYCODE_A	29	字母 A 键
KEYCODE_B	30	字母 B 键
KEYCODE_C	31	字母 C 键
KEYCODE_D	32	字母 D 键
KEYCODE_E	33	字母 E 键
KEYCODE_F	34	字母 F 键
KEYCODE_G	35	字母 G 键
KEYCODE_H	36	字母 H 键
KEYCODE_I	37	字母 I 键
KEYCODE_J	38	字母 J 键
KEYCODE_K	39	字母 K 键
KEYCODE_L	40	字母 L 键
KEYCODE_M	41	字母 M 键
KEYCODE_N	42	字母 N 键
KEYCODE_O	43	字母 O 键
KEYCODE_P	44	字母 P 键
KEYCODE_Q	45	字母 Q 键
KEYCODE_R	46	字母 R 键
KEYCODE_S	47	字母 S 键
KEYCODE_T	48	字母 T 键
KEYCODE_U	49	字母 U 键

变量名	键值	备注
KEYCODE_V	50	字母 V 键
KEYCODE_W	51	字母 W 键
KEYCODE_X	52	字母 X 键
KEYCODE_Y	53	字母 Y 键
KEYCODE_Z	54	字母 Z 键
KEYCODE_COMMA	55	, 键
KEYCODE_PERIOD	56	. 键
KEYCODE_ALT_LEFT	57	左 ALT 键
KEYCODE_ALT_RIGHT	58	右 ALT 键
KEYCODE_SHIFT_LEFT	59	左 SHIFT 键
KEYCODE_SHIFT_RIGHT	60	右 SHIFT 键
KEYCODE_TAB	61	TAB 键
KEYCODE_SPACE	62	空格键
KEYCODE_SYM	63	符号键
KEYCODE_EXPLORER	64	Explorer 键
KEYCODE_ENVELOPE	65	Envelope 键
KEYCODE_ENTER	66	回车键
KEYCODE_DEL	67	DEL 键
KEYCODE_GRAVE	68	` 键
KEYCODE_MINUS	69	- 键
KEYCODE_EQUALS	70	= 键
KEYCODE_LEFT_BRACKET	71	[键
KEYCODE_RIGHT_BRACKET	72	] 键
KEYCODE_BACKSLASH	73	\ 键
KEYCODE_SEMICOLON	74	; 键

变量名	键值	备注
KEYCODE_APOSTROPHE	75	‘ 键
KEYCODE_SLASH	76	/ 键
KEYCODE_AT	77	@ 键
KEYCODE_NUM	78	Number 键
KEYCODE_HEADSETHOOK	79	Headset Hook 键
KEYCODE_FOCUS	80	Camera Focus 键
KEYCODE_PLUS	81	+ 键
KEYCODE_MENU	82	菜单键
KEYCODE_NOTIFICATION	83	Notification 键
KEYCODE_SEARCH	84	Search 键
KEYCODE_MEDIA_PLAY_PAUSE	85	Play/Pause media 键
KEYCODE_MEDIA_STOP	86	Stop media 键
KEYCODE_MEDIA_NEXT	87	Play Next media 键
KEYCODE_MEDIA_PREVIOUS	88	Play Previous media 键
KEYCODE_MEDIA_REWIND	89	Rewind media 键
KEYCODE_MEDIA_FAST_FORWARD	90	Fast Forward media 键
KEYCODE_MUTE	91	Mute 键
KEYCODE_PAGE_UP	92	Page Up 键
KEYCODE_PAGE_DOWN	93	Page Down 键
KEYCODE_PICTSYMBOLS	94	Picture Symbols 键
KEYCODE_SWITCH_CHARSET	95	Switch Charset 键
KEYCODE_BUTTON_A	96	A Button 键
KEYCODE_BUTTON_B	97	B Button 键
KEYCODE_BUTTON_C	98	C Button 键

变量名	键值	备注
KEYCODE_BUTTON_X	99	X Button 键
KEYCODE_BUTTON_Y	100	Y Button 键
KEYCODE_BUTTON_Z	101	Z Button 键
KEYCODE_BUTTON_L1	102	L1 Button 键
KEYCODE_BUTTON_R1	103	R1 Button 键
KEYCODE_BUTTON_L2	104	L2 Button 键
KEYCODE_BUTTON_R2	105	R2 Button 键
KEYCODE_BUTTON_THUMBL	106	Left Thumb Button 键
KEYCODE_BUTTON_THUMBR	107	Right Thumb Button 键
KEYCODE_BUTTON_START	108	Start Button 键
KEYCODE_BUTTON_SELECT	109	Select Button 键
KEYCODE_BUTTON_MODE	110	Mode Button 键
KEYCODE_ESCAPE	111	Escape 键
KEYCODE_FORWARD_DEL	112	Forward Delete 键
KEYCODE_CTRL_LEFT	113	Left Control 键
KEYCODE_CTRL_RIGHT	114	Right Control 键
KEYCODE_CAPS_LOCK	115	Caps Lock 键
KEYCODE_SCROLL_LOCK	116	Scroll Lock 键
KEYCODE_META_LEFT	117	Left Meta 键
KEYCODE_META_RIGHT	118	Right Meta 键
KEYCODE_FUNCTION	119	Function 键
KEYCODE_SYSRQ	120	System Request / Print Screen 键
KEYCODE_BREAK	121	Break / Pause 键

变量名	键值	备注
KEYCODE_MOVE_HOME	122	Home Movement 键
KEYCODE_MOVE_END	123	End Movement 键
KEYCODE_INSERT	124	Insert 键
KEYCODE_FORWARD	125	Forward 键
KEYCODE_MEDIA_PLAY	126	Play media 键
KEYCODE_MEDIA_PAUSE	127	Pause media 键
KEYCODE_MEDIA_CLOSE	128	Close media 键
KEYCODE_MEDIA_EJECT	129	Eject media 键
KEYCODE_MEDIA_RECORD	130	Record media 键
KEYCODE_F1	131	F1 键
KEYCODE_F2	132	F2 键
KEYCODE_F3	133	F3 键
KEYCODE_F4	134	F4 键
KEYCODE_F5	135	F5 键
KEYCODE_F6	136	F6 键
KEYCODE_F7	137	F7 键
KEYCODE_F8	138	F8 键
KEYCODE_F9	139	F9 键
KEYCODE_F10	140	F10 键
KEYCODE_F11	141	F11 键
KEYCODE_F12	142	F12 键

变量名	键值	备注
KEYCODE_NUM_LOCK	143	Num Lock 键
KEYCODE_NUMPAD_0	144	Numeric keypad ‘0' 键
KEYCODE_NUMPAD_1	145	Numeric keypad '1' 键
KEYCODE_NUMPAD_2	146	Numeric keypad '2' 键
KEYCODE_NUMPAD_3	147	Numeric keypad '3' 键
KEYCODE_NUMPAD_4	148	Numeric keypad '4' 键
KEYCODE_NUMPAD_5	149	Numeric keypad '5' 键
KEYCODE_NUMPAD_6	150	Numeric keypad '6' 键
KEYCODE_NUMPAD_7	151	Numeric keypad '7' 键
KEYCODE_NUMPAD_8	152	Numeric keypad '8' 键
KEYCODE_NUMPAD_9	153	Numeric keypad '9' 键
KEYCODE_NUMPAD_DIVIDE	154	Numeric keypad '/' 键
KEYCODE_NUMPAD_MULTIPLY	155	Numeric keypad '*' 键
KEYCODE_NUMPAD_SUBTRACT	156	Numeric keypad '-' 键
KEYCODE_NUMPAD_ADD	157	Numeric keypad '+' 键
KEYCODE_NUMPAD_DOT	158	Numeric keypad '.' 键
KEYCODE_NUMPAD_COMMA	159	Numeric keypad ',' 键
KEYCODE_NUMPAD_ENTER	160	Numeric keypad Enter 键
KEYCODE_NUMPAD_EQUALS	161	Numeric keypad '=' 键
KEYCODE_NUMPAD_LEFT_PAREN	162	Numeric keypad '(' 键
KEYCODE_NUMPAD_RIGHT_PAREN	163	Numeric keypad ')' 键

变量名	键值	备注
KEYCODE_VOLUME_MUTE	164	Volume Mute 键
KEYCODE_INFO	165	Info 键
KEYCODE_CHANNEL_UP	166	Channel up 键
KEYCODE_CHANNEL_DOWN	167	Channel down 键
KEYCODE_ZOOM_IN	168	Zoom in 键
KEYCODE_ZOOM_OUT	169	Zoom out 键
KEYCODE_TV	170	TV 键
KEYCODE_WINDOW	171	Window 键
KEYCODE_GUIDE	172	Guide 键
KEYCODE_DVR	173	DVR 键
KEYCODE_BOOKMARK	174	Bookmark 键
KEYCODE_CAPTIONS	175	Toggle captions 键
KEYCODE_SETTINGS	176	Settings 键
KEYCODE_TV_POWER	177	V power 键
KEYCODE_TV_INPUT	178	TV input 键
KEYCODE_STB_POWER	179	Set-top-box power 键
KEYCODE_STB_INPUT	180	Set-top-box input 键
KEYCODE_AVR_POWER	181	A/V Receiver power 键
KEYCODE_AVR_INPUT	182	A/V Receiver input 键
KEYCODE_PROG_RED	183	Red “programmable" 键
KEYCODE_PROG_GREEN	184	Green "programmable" 键

变量名	键值	备注
KEYCODE_PROG_YELLOW	185	Yellow “programmable" 键
KEYCODE_PROG_BLUE	186	Blue "programmable" 键
KEYCODE_APP_SWITCH	187	App switch 键
KEYCODE_BUTTON_1	188	Generic Game Pad Button #1
KEYCODE_BUTTON_2	189	Generic Game Pad Button #2
KEYCODE_BUTTON_3	190	Generic Game Pad Button #3
KEYCODE_BUTTON_4	191	Generic Game Pad Button #4
KEYCODE_BUTTON_5	192	Generic Game Pad Button #5
KEYCODE_BUTTON_6	193	Generic Game Pad Button #6
KEYCODE_BUTTON_7	194	Generic Game Pad Button #7
KEYCODE_BUTTON_8	195	Generic Game Pad Button #8
KEYCODE_BUTTON_9	196	Generic Game Pad Button #9
KEYCODE_BUTTON_10	197	Generic Game Pad Button #10
KEYCODE_BUTTON_11	198	Generic Game Pad Button #11
KEYCODE_BUTTON_12	199	Generic Game Pad Button #12
KEYCODE_BUTTON_13	200	Generic Game Pad Button #13
KEYCODE_BUTTON_14	201	Generic Game Pad Button #14
KEYCODE_BUTTON_15	202	Generic Game Pad Button #15
KEYCODE_BUTTON_16	203	Generic Game Pad Button #16
KEYCODE_LANGUAGE_SWITCH	204	Language Switch 键
KEYCODE_MANNER_MODE	205	Manner Mode 键
KEYCODE_3D_MODE	206	3D Mode 键
KEYCODE_CONTACTS	207	Contacts special function 键

变量名	键值	备注
KEYCODE_CALENDAR	208	Calendar special function 键
KEYCODE_MUSIC	209	Music special function 键
KEYCODE_CALCULATOR	210	Calculator special function 键
KEYCODE_ZENKAKU_HANKAKU	211	Japanese full-width / half-width key
KEYCODE_EISU	212	Japanese alphanumeric key
KEYCODE_MUHENKAN	213	Japanese non-conversion key
KEYCODE_HENKAN	214	Japanese conversion key
KEYCODE_KATAKANA_HIRAGANA	215	Japanese katakana / hiragana key
KEYCODE_YEN	216	Japanese Yen key
KEYCODE_RO	217	Japanese Ro key
KEYCODE_KANA	218	Japanese kana key
KEYCODE_ASSIST	219	Assist key. Launches the global assist activity
KEYCODE_BRIGHTNESS_DOWN	220	Brightness Down key
KEYCODE_BRIGHTNESS_UP	221	Brightness Up key

7. 接入技术规范和验收注意事项

1. 接入技术规范

1、 阿里 TV 外设规范包含：技术文档中指定的遥控器和手柄键值；体感、震动、加速度通信格式等技术标准；
2、 应用运行全场景保障手柄、遥控器能同时兼容和使用；体感、震动、加速度等在
3、 应用运行全场景保障手柄、遥控器的[HOME]键点击回到桌面；
4、 应用运行全场景保障手柄、遥控器的[BACK]键点击可后退和正常路径的应用退出；
5、 游戏结束退出到 YUNOS 系统界面，禁止应用后台进程抢占、拦截操控器指令等任何影响系统正常运行的行为；
6、 应用接入淘宝 ID、支付宝 SDK 等相关阿里服务程序和代码，需保障手柄和遥控器的正常操作；
7、 应用接入阿里 TV 游戏平台必须遵循外设 SDK 和技术标准，不得自行规定外设适配的型号；不得有任何违反平台适配标准的排他行为；
8、 请应用适配完成后自测并填写自测试验收表，进入平台验收的上线流程；

2. 应用接入外设兼容自测试验收表

功能和行为描述	YUNOS 系统 （正常√ 异常×）	应用 （正常√ 异常×）	外设操控 （正常√ 异常×）
应用启动			
应用全进程			
应用后台进程			
应用退出功能			
淘宝 ID 运行			
支付 SDK 运行			
运营 SDK 运行			
平台外设兼容			