手表按键定义及接入说明

表环物理按键功能定义需求文档 v2.0

1.硬件形式：

上键/表冠

旋转触发旋钮事件

按键定义为功能键，发送KEYCODE_STEM_PRIMARY事件

下键

电源键，发送KEYCODE_POWER事件

2.国内版本物理按键新方案(202505)

20250514新方案按键应用定制能力

20250514按键新方案按键应用定制能力				
上键	**KeyCode**	**STEM_PRIMARY**	**单击F1, 双击F2**	**只有F1**
	延迟	无延迟	有300ms延迟	无延迟
	单击	应用/表盘	应用/表盘	应用/表盘
	双击	应用/多任务	应用/多任务	无
	长按1秒	应用/小爱	小爱	小爱
	>300ms的任意长按	应用	无	无
下键	**KeyCode**	**POWER**		
	单击	应用/控制中心		
	双击	应用/无系统操作		
	三击	应用/紧急呼叫		
	长按3秒	关机菜单		
	>300ms && < 3s的任意长按	应用		

表格中, 单击, 双击, 长按等系统默认操作, 目前是功能固定的, 后期如果有需要可以修改成用户可配置.

2.1 上按键处理(KEYCODE_STEM_PRIMARY):

2.1.1 默认情况, 应用不加入名单,不设置特定window tag

单击会返回表盘,双击拉起多任务界面

2.1.2 需要即时响应的情况

应用截获KEYCODE_STEM_PRIMARY, 需要应用内处理单击双击长按等事件, 此为wearos标准操作可以参考google官方文档.

需要注意的是,一旦应用配置了截获此按键,如果应用在某些条件下不处理此按键,需要在onKeyDown处理中返回false,框架才会执行系统默认操作,如单击返回表盘,双击打开多任务.

在onKeyDown中返回true, 系统才不会执行默认操作.

> 📌 示例应用:
>
> 闹钟响起时,按上键,应用在onKeyDown中判断到KEYCODE_STEM_PRIMARY后执行"十分钟后提醒"
>
> 这是闹钟目前代码,其实闹钟可以改为"**框架把上按键转换成F1和F2,要求应用允许300ms延时收到按键事件**"处理

如果应用内没有**长按**需求, 可以让框架处理**长按1s打开语音助手**的系统操作,两种方法:

i. 根据需要将package name或者activity name给到框架,加入名单中

ii. 设置window tag中flag为 *FLAG_USE_DEFAULT_STEM_KEY_LONG_PRESS, 设置方式参考* *手表按键定义及接入说明*

2.1.3 框架把上按键转换成F1和F2, 收到事件有300ms延迟

框架把单击上按键转换成KEYCODE_F1,双击上按键转换成KEYCODE_F2,省去了应用内判断单双击的麻烦,同时方便处理回表盘和拉起多任务的系统操作

应用根据需要在收到KEYCODE_F1事件时onKeyDown中return true 或者 return false, 如果返回true,系统不再处理,返回false,系统默认退出应用并返回表盘.

应用根据需要在收到KEYCODE_F2事件时onKeyDown中return true 或者 return false, 如果返回true,系统不再处理,返回false,系统默认拉起多任务界面.

两种配置方法:

i. 根据需要将package name或者activity name给到框架,加入名单中

ii. 设置window tag中flag为 *FLAG_CONVERT_STEM_TO_FX, 设置方式参考 手表按键定义及接入说明*

📌 示例应用:

秒表, 在onKeyUp中判断到KEYCODE_F1执行开始/暂停操作并return true,收到KEYCODE_F2

(秒表 开始/暂停 有延迟没关系,只要开始和暂停有同样时长的延时就可以.但是按电源键插入计次数据不能有延迟)

2.1.4 框架把点击上键无300ms延迟转换成F1

框架把点击上键只会转换成F1, 多次点击发送多次F1给应用, 用于应用只处理单击的情况, 并且无延迟收到.

此时系统双击上键打开多任务功能被屏蔽, 长按打开语音助手仍然生效, 使用此功能需要和产品确认需求

应用根据需要在收到KEYCODE_F1事件时onKeyDown中return true 或者 return false, 如果返回true,系统不再处理,返回false,系统默认退出应用并返回表盘.

两种配置方法:

i. 根据需要将package name或者activity name给到框架,加入名单中

ii. 设置window tag中flag为 ***FLAG_CONVERT_STEM_TO_FX*** | ***FLAG_CONVERT_STEM_TO_F1_ONLY***

要求两个flag组合设置置

2.1.5 应用需要屏蔽上按键

特定应用或者应用内特定窗口,需要屏蔽上按键,应用不处理, 不需要回表盘,不拉起多任务,三种方法:

i. 根据需要将package name或者activity name给到框架,加入名单中

ii. 应用在onKeyDown中return true

iii. 设置window tag中flag为 *FLAG_IGNORE_STEM_KEY, 设置方式参考 手表按键定义及接入说明*

📌 示例应用:

配对向导界面屏蔽上按键(框架已将配对向导package name加入名单中)

2.2 下按键处理(KEYCODE_POWER):

默认情况下应用不会收到KEYCODE_POWER事件, 系统会执行默认操作,如单击打开控制中心,三击打开紧急呼叫

应用可以通过配置特定window tag来开启收到KEYCODE_POWER按键事件

2.2.1 如果需要自己处理电源按键并且屏蔽系统默认单击和三击的系统操作,需要同时配置以下两个:

i. 设置window tag中flag为 *FLAG_USE_POWER_KEY, 设置方式参考 手表按键定义及接入说明*

ii. onKeyDown中return true, 如果不override onKeyDown或者return false, 则应用在处理完onKeyDown之后还会执行系统默认操作

📌 示例应用:

秒表,单击电源键添加一条计次记录

2.2.2 特定应用或者应用内特定窗口,需要屏蔽下按键, 不打开控制中心, 不打开紧急呼叫,三种方法:

i. 根据需要将package name或者activity name给到框架,加入名单中

ii. 应用在onKeyDown中return true

iii. 设置window tag中flag为 *FLAG_IGNORE_POWER_KEY, 设置方式参考 手表按键定义及接入说明*

📌 示例应用:

配对向导界面屏蔽下按键(框架已将配对向导package name加入名单中)

设置window tag:

代码块

```
public class TestActivity extends Activity {


    public static final int FLAG_NONE = 0x00000000;
    public static final int FLAG_USE_POWER_KEY = 0x00000001;
    public static final int FLAG_CONVERT_STEM_TO_FX = 0x00000002;

    //FLAG_IGNORE_STEM_KEY 与 FLAG_CONVERT_STEM_TO_FX 和 FLAG_USE_DEFAULT_STEM_KEY_LONG_PRESS
    //不兼容,如果同时设置,只有FLAG_IGNORE_STEM_KEY生效
    public static final int FLAG_IGNORE_STEM_KEY = 0x00000004;
    //屏蔽系统的单击打开控制中心和三击打开紧急呼叫,但是长按3S打开关机菜单仍然生效
    //FLAG_IGNORE_POWER_KEY 与 FLAG_USE_POWER_KEY 不兼容, 同时设置,只有FLAG_IGNORE_POWER_KEY生效
    public static final int FLAG_IGNORE_POWER_KEY = 0x00000008;
    public static final int FLAG_USE_DEFAULT_STEM_KEY_LONG_PRESS = 0x00000010;

    //需要和 FLAG_CONVERT_STEM_TO_FX 一起使用
    //设置此flag后, 点击上按键只会收到F1, 无300ms延迟
    public static final int FLAG_CONVERT_STEM_TO_F1_ONLY = 0x00000020;


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

    private void startSomeWorking() {
        //...
        setWindowFlag(getWindow(), FLAG_USE_POWER_KEY | FLAG_CONVERT_STEM_TO_FX);
    }

    private void stopSomeWorking() {
        //...
        setWindowFlag(getWindow(), FLAG_NONE);
    }

```

```
38      //声明此应用接收power key，并且要求上键单击转换成KEYCODE_F1，双击转换成
    FKEYCODE_F2上报给应用
39      public void setWindowFlag(Window window, int flags) {
40          WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
41          layoutParams.setTitle(flagsConvertToWinTag(flags));
42          window.setAttributes(layoutParams);
43      }
44
45
46      //tag 必须以"miwear_"为前缀
47      public String flagsConvertToWinTag(int flags) {
48          return "miwear_" + String.valueOf(flags);
49      }
50
51      public int tagConvertToFlags(String tag) {
52          if (tag.startsWith("miwear_")) {
53              try {
54                  return Integer.parseInt(tag.substring("miwear_".length()));
55              } catch (NumberFormatException e) {
56                  return 0;
57              }
58          }
59          return 0;
60      }
61
62  }
```

3.应用对按键/旋钮的事件处理

应用只能对功能键和旋钮事件进行处理，电源键无法拦截

功能键

KEYCODE_STEM_PRIMARY

系统对*KEYCODE_STEM_PRIMARY*功能键有默认定义，如果应用要自定义该按键处理方式，需要声明权限 "android.permission.OVERRIDE_SYSTEM_KEY_BEHAVIOR_IN_FOCUSED_WINDOW"，此权限定义在frameworks/base/core/res/AndroidManifest.xml

代码块

```
1       <!-- @SystemApi Allows focused window to override the default behavior of
    supported system keys.
2           The following keycodes are supported:
```

```
         <p> KEYCODE_STEM_PRIMARY
         <p>If an app is granted this permission and has a focused window, it will be allowed to
            receive supported key events that are otherwise handled by the system. The app can choose
            to consume the key events and trigger its own behavior, in which case the default key
            behavior will be skipped.
         <p>For example, KEYCODE_STEM_PRIMARY by default opens recent app launcher. If the foreground
            fitness app is granted this permission, it can repurpose the KEYCODE_STEM_PRIMARY button
            to pause/resume the current fitness session.
         <p>Protection level: signature|privileged

         @FlaggedApi("com.android.input.flags.override_key_behavior_permission_apis")
         @hide -->
    <permission android:name="android.permission.OVERRIDE_SYSTEM_KEY_BEHAVIOR_IN_FOCUSED_WINDOW"
         android:protectionLevel="signature|privileged" />
```

此权限需要系统签名或者是特权应用(system|system_ext|vendor|odm|product/priv-app/目录下)，如果是特权应用且不是系统签名，还要在系统源码中添加privapp-permissions声明文件并编译到系统中，声明方式可参考frameworks/base/data/etc/privapp-permissions-platform.xml

代码块

```
<permissions>
    <privapp-permissions package="com.android.launcher3">
        <permission name="android.permission.WRITE_SECURE_SETTINGS"/>
        <permission name="android.permission.READ_SYSTEM_GRAMMATICAL_GENDER"/>
        <permission name="android.permission.OVERRIDE_SYSTEM_KEY_BEHAVIOR_IN_FOCUSED_WINDOW"/>
    </privapp-permissions>
</permissions>
```

应用处理功能键示例：

代码块

```
//声明权限
```

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

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_STEM_PRIMARY) {
        if (mInterceptCount < 10) {
            mInterceptCount++;
            //应用消费掉，不让系统处理
            return true;
        } else {
            mInterceptCount = 0;
        }
    }
    return super.onKeyDown(keyCode, event);
}

```

旋钮

处理旋钮事件

代码块

```
@Override
public boolean onGenericMotionEvent(MotionEvent event) {
    if (event.isFromSource(InputDevice.SOURCE_ROTARY_ENCODER)) {
        float scroll = event.getAxisValue(MotionEvent.AXIS_SCROLL);
        //正对表盘顺时针旋转为负值，逆时针旋转为正值
        mRootScrollView.scrollBy(0, (int)(scroll * 5));
    }
    return super.onGenericMotionEvent(event);
}
```

4.系统集成

按键

按键映射

device/xiaomi/common/gpio-keys.kl

代码块

```
key 115   STEM_PRIMARY
key 116   POWER
```

旋钮

功能集成参考：

https://gerrit.odm.mioffice.cn/c/vendor/xiaomi/aurora/+/603156
https://gerrit.odm.mioffice.cn/c/device/xiaomi/aurora/+/603161

配置转动旋钮是否可以唤醒屏幕：

frameworks/opt/wear/res/values/config.xml

代码块

```
<!-- Whether to enable rotary encoder based wakeups. -->
<bool name="config_rotaryEncoderWakeEnabled">true</bool>
```