

Unisoc Confidential For hiar

Android 10.0 拨号盘应用客制化指导手册

文档版本
发布日期

V1.0
2020-10-09

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档对 Android 10.0 拨号盘模块及常见的客制化问题进行介绍。

读者对象


本文档主要适用于与 Android 10.0 拨号盘模块相关的开发和测试人员。

缩略语

缩略语	英文全名	中文解释
ANR	Android Not Responding	应用程序无响应
SIM	Subscriber Identity Module	用户识别模块

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-10-09	第一次正式发布。

关键字

拨号盘、编译、客制化。

目 录

1 拨号盘模块介绍.....	1
1.1 代码目录.....	1
1.2 编译方法.....	1
2 常见客制化问题.....	2
2.1 合并俄罗斯号码.....	2
2.1.1 问题描述.....	2
2.1.2 解决方案.....	2
2.1.3 编译模块.....	2
2.1.4 验证方式.....	2
2.2 修改未接来电显示记录上限设置.....	2
2.2.1 问题描述.....	2
2.2.2 解决方案.....	3
2.2.3 编译模块.....	3
2.2.4 验证方式.....	3
2.3 修改双卡运营商显示.....	3
2.3.1 问题描述.....	3
2.3.2 解决方案.....	3
2.3.3 编译模块.....	4
2.3.4 验证方式.....	4
2.4 预置通话记录资料.....	5
2.4.1 问题描述.....	5
2.4.2 解决方案.....	5
2.4.3 编译模块.....	9
2.4.4 验证方式.....	9
2.5 修改默认 Tab 界面显示.....	9
2.5.1 问题描述.....	9
2.5.2 解决方案.....	9
2.5.3 编译模块.....	10
2.5.4 验证方式.....	10

1 拨号盘模块介绍

拨号盘是 Android 手机系统必不可少的组成部分，主要提供拨打电话、显示通话记录、收藏联系人等功能。拨号盘与联系人模块配合使用，可以对联系人和通话记录进行管理。

1.1 代码目录

Android 10.0 中拨号盘模块的代码目录：`/packages/apps/Dialer/`

1.2 编译方法

在项目主目录下，参考以下步骤进行编译：

步骤 1 输入 `source/build/envsetup.sh`，将编译脚本加载到内存。

步骤 2 输入 `lunch`，然后选择具体的手机型号，如 `s9863a10c10_Natv-userdebug-native`。

步骤 3 输入 `make <xxxx>` 开始编译。`<xxxx>` 代表模块名称，如 `Dialer`、`SystemUI` 等。

---结束

2 常见客制化问题

2.1 合并俄罗斯号码

2.1.1 问题描述

在拨号盘的通话记录和搜索功能中，将以+7 和 8 开头的俄罗斯号码合并。

2.1.2 解决方案

在对应的 xml 文件中分别增加 “+7” 和 “8” 的 item，将+7 和 8 开头的俄罗斯号码进行合并。

Android 10.0 已实现此功能，类似需求可按如下方式进行修改。

- 代码路径：
/packages/apps/Dialer/java/com/android/dialer/app/res/values/strings.xml
- 参考修改：

```
<string-array name="country_code_with_plus">
    <item>+7</item>
</string-array>
<string-array name="country_code_no_plus">
    <item>8</item>
</string-array>
```

2.1.3 编译模块

编译模块：Dialer。具体的编译方法参见“1.2 编译方法”。

2.1.4 验证方式

拨打号码+7123456，然后可以在通话记录中搜索到号码为 8123456 的联系人。

2.2 修改未接来电显示记录上限设置

2.2.1 问题描述

目前锁屏界面最多只能显示 8 条未接来电，要求修改未接来电显示记录的上限。

2.2.2 解决方案

修改 MAX_NUMBER_OF_MISSED_CALL 和 MAX_NOTIFICATIONS_PER_TAG 的值，使得在锁屏界面能够显示更多的未接来电记录。

说明

显示过多的未接来电记录可能会引起 ANR (Android Not Responding, 应用程序无响应)，请酌情修改。

修改 MAX_NUMBER_OF_MISSED_CALL 的值

- 代码路径：
frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/notification/stack/NotificationChildrenContainer.java

- 参考修改：

```
private static final int MAX_NUMBER_OF_MISSED_CALL = 10;
```

修改 MAX_NOTIFICATIONS_PER_TAG 的值

- 代码路径：
/packages/apps/Dialer/java/com/android/dialer/notification/NotificationThrottler.java
- 参考修改：

```
private static final int MAX_NOTIFICATIONS_PER_TAG = 10;
```

2.2.3 编译模块

编译模块：Dialer、SystemUI。具体的编译方法参见“1.2 编译方法”。

说明

SystemUI 模块的代码目录：/frameworks/base/packages/SystemUI

2.2.4 验证方式

拨打修改后的手机，查看锁屏状态下未接来电显示记录是否能达到修改后的数量。

2.3 修改双卡运营商显示

2.3.1 问题描述

将拨号盘中所有关于 SIM1，SIM2 的显示，如来电或拨号时，统一显示为 SIM+运营商。

2.3.2 解决方案

分别修改通话详情界面和来电界面的 SIM 显示，将关于 SIM1，SIM2 的显示统一显示为 SIM+运营商。

修改通话详情界面

- 代码修改路径：/packages/apps/Dialer/
- 参考修改：


```
diff --git a/java/com/android/dialer/app/callog/PhoneCallDetailsHelper.java
b/java/com/android/dialer/app/callog/PhoneCallDetailsHelper.java
index 5c81427..06a9f09 100644
--- a/java/com/android/dialer/app/callog/PhoneCallDetailsHelper.java
+++ b/java/com/android/dialer/app/callog/PhoneCallDetailsHelper.java
@@ -264,7 +264,7 @@ public class PhoneCallDetailsHelper
    .getActiveSubscriptionInfo(context, i, false);
    if (subscriptionInfo != null
        && iccId.equals(subscriptionInfo.getIccId())) {
-        simIdentification = resources.getString(R.string.sim) + (i + 1);
+        simIdentification = resources.getString(R.string.sim);
    }
}
```

修改来电显示界面

- 代码路径：/packages/apps/Dialer/
- 参考修改：

```
diff --git a/java/com/android/incallui/incall/impl/res/values/strings.xml
b/java/com/android/incallui/incall/impl/res/values/strings.xml
index 9c78408..05da0c4 100644
--- a/java/com/android/incallui/incall/impl/res/values/strings.xml
+++ b/java/com/android/incallui/incall/impl/res/values/strings.xml
@@ -73,7 +73,7 @@
    [CHAR LIMIT=32] -->
    <string name="incall_note_sent">Note sent</string>
-   <string name="xliff_string1">SIM1</string>
-   <string name="xliff_string2">SIM2</string>
+   <string name="xliff_string1">SIM</string>
+   <string name="xliff_string2">SIM</string>
</resources>
```

2.3.3 编译模块

编译模块：Dialer。具体的编译方法参见“1.2 编译方法”。

2.3.4 验证方式

拨打电话，查看是否变成 SIM+运营商。

2.4 预置通话记录资料

2.4.1 问题描述

在 demo 版本中预置通话记录资料。

2.4.2 解决方案

先在 AndroidManifest.xml 文件中配置开机广播，然后新增接收广播并判断是否为烧机后的第一次开机。如果是烧机后第一次开机就新增 Service 文件并预置通话记录。

配置开机广播

- 代码路径：/packages/apps/Dialer/
- 参考修改：

```
diff --git a/java/com/android/dialer/app/AndroidManifest.xml b/java/com/android/dialer/app/AndroidManifest.xml
index 62c712e..a1fe01c 100644
--- a/java/com/android/dialer/app/AndroidManifest.xml
+++ b/java/com/android/dialer/app/AndroidManifest.xml
@@ -280,6 +280,15 @@
     </intent-filter>
     </receiver>
     <!-- @} -->
+    <receiver android:name="com.android.dialer.app.calllog.PreDefineCalllogReceiver">
+        <intent-filter>
+            <action android:name="android.intent.action.BOOT_COMPLETED" />
+        </intent-filter>
+    </receiver>
+
+    <service android:name="com.android.dialer.app.calllog.PreDefineCalllogService"
+        android:exported="false">
+    </service>

 </application>
</manifest>
```

新增接收广播和 Service 文件

- 代码路径：/packages/apps/Dialer/java/com/android/dialer/app/calllog/
- 参考修改：新增 PreDefineCalllogReceiver.java 文件，PreDefineCalllogReceiver.java 文件代码如下。

```
package com.android.dialer.app.calllog;

import android.util.Log;
import android.app.AppGlobals;
```

```
import android.content.Context;
import android.content.Intent;
import android.content.BroadcastReceiver;
import android.os.RemoteException;

public class PreDefineCalllogReceiver extends BroadcastReceiver {

    private static final String TAG = "PreDefineCalllogReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        boolean isFirstBoot = false;
        try {
            isFirstBoot = AppGlobals.getPackageManager().isFirstBoot();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        Log.d(TAG, "isFirstBoot : " + isFirstBoot);
        if (isFirstBoot) {
            context.startService(new Intent(context, PreDefineCalllogService.class));
        }
    }
}
```

在 Service 文件中预置通话记录

- 代码路径: /packages/apps/Dialer/java/com/android/dialer/app/calllog/
- 参考修改: 新增 PreDefineCalllogService.java 文件, PreDefineCalllogService.java 文件代码如下。

```
package com.android.dialer.app.calllog;

import android.app.IntentService;
import android.content.Intent;
import android.provider.CallLog;
import android.provider.CallLog.Calls;
import android.util.Log;
import android.content.Context;

import java.util.Random;

import android.os.RemoteException;
import android.provider.CallLog.Calls;
import android.util.Log;
```

```
import android.content.ContentProviderClient;
import android.content.ContentValues;
import android.os.AsyncTask;

public class PreDefineCalllogService extends IntentService {
    private static final String TAG = "PreDefineCalllogService";

    private static final int[] CALL_TYPES = new int[] {
        Calls.INCOMING_TYPE, Calls.OUTGOING_TYPE, Calls.MISSED_TYPE,
    };
    private static final Random RNG = new Random();

    public PreDefineCalllogService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {

        int count = 3;
        ContentValues[] values = new ContentValues[count];
        for (int i = 0; i < count; i++) {
            values[i] = new ContentValues();
            values[i].put(Calls.NUMBER, generateRandomNumber());
            values[i].put(Calls.NUMBER_PRESENTATION, Calls.PRESENTATION_ALLOWED);
            values[i].put(Calls.DATE, System.currentTimeMillis());
            values[i].put(Calls.DURATION, RNG.nextInt(200));
        }
        new AsyncCallLogInserter(values).execute(new Void[0]);
    }

    private static String generateRandomNumber() {
        return String.format("%09d", RNG.nextInt(1000000000));
    }

    /**
     * Inserts a given number of entries in the call log based on the values given.
     */
    private final class AsyncCallLogInserter extends AsyncTask<Void, Integer, Integer> {
        /**
```

```
* The number of items to insert.
*/

private final ContentValues[] mValues;

public AsyncCallLogInserter(ContentValues[] values) {
    mValues = values;
}

protected Integer doInBackground(Void... params) {
    Log.d(TAG, "doInBackground");
    return insertIntoCallLog();
}

/**
 * Inserts a number of entries in the call log based on the given templates.
 *
 * @return the number of inserted entries
 */
private Integer insertIntoCallLog() {
    int inserted = 0;

    for (int index = 0; index < mValues.length; ++index) {
        ContentValues values = mValues[index];
        // These should not be set.
        values.putNull(Calls._ID);
        // Add some randomness to the date. For each new entry being added, add an extra
        // day to the maximum possible offset from the original.
        // values.put(Calls.DATE,
        // values.getAsLong(Calls.DATE)
        // - RNG.nextInt(24 * 60 * 60 * (index + 1)) * 1000L);
        // Add some randomness to the duration.
        if (values.getAsLong(Calls.DURATION) > 0) {
            values.put(Calls.DURATION, RNG.nextInt(30 * 60 * 60 * 1000));
        }

        // Overwrite type.
        values.put(Calls.TYPE, CALL_TYPES[RNG.nextInt(CALL_TYPES.length)]);

        // Clear cached columns.
        values.putNull(Calls.CACHED_FORMATTED_NUMBER);
    }
}
```

```
values.putNull(Calls.CACHED_LOOKUP_URI);
values.putNull(Calls.CACHED_MATCHED_NUMBER);
values.putNull(Calls.CACHED_NAME);
values.putNull(Calls.CACHED_NORMALIZED_NUMBER);
values.putNull(Calls.CACHED_NUMBER_LABEL);
values.putNull(Calls.CACHED_NUMBER_TYPE);
values.putNull(Calls.CACHED_PHOTO_ID);

// Insert into the call log the newly generated entry.
ContentProviderClient contentProvider =
    getContentResolver().acquireContentProviderClient(
        Calls.CONTENT_URI);
try {
    Log.d(TAG, "adding entry to call log");
    contentProvider.insert(Calls.CONTENT_URI, values);
    ++inserted;
    this.publishProgress(inserted);
} catch (RemoteException e) {
    Log.d(TAG, "insert failed", e);
}
}
return inserted;
}
```

2.4.3 编译模块

编译模块：Dialer。具体的编译方法参见“1.2 编译方法”。

2.4.4 验证方式

烧机开机以后，查看是否有默认的通话记录。

2.5 修改默认 Tab 界面显示

2.5.1 问题描述

将第一次加载 Tab 界面时默认显示的 Favorite Tab 界面修改为通话记录 Tab 界面。

2.5.2 解决方案

修改第一次进入拨号盘默认加载界面为通话记录 Tab 界面。

- 代码路径: /packages/apps/Dialer/
- 参考修改:

```
diff --git a/java/com/android/dialer/main/impl/OldMainActivityPeer.java
b/java/com/android/dialer/main/impl/OldMainActivityPeer.java
index cff6743..3909b56 100644
--- a/java/com/android/dialer/main/impl/OldMainActivityPeer.java
+++ b/java/com/android/dialer/main/impl/OldMainActivityPeer.java
@@ -1769,16 +1769,16 @@ public class OldMainActivityPeer implements MainActivityPeer, FragmentUtilListen
    */
    @TabIndex
    int getLastTab() {
-        @TabIndex int tabIndex = TabIndex.SPEED_DIAL;
+        @TabIndex int tabIndex = TabIndex.CALL_LOG;
        tabIndex =
            StorageComponent.get(context)
                .unencryptedSharedPreferences()
-            .getInt(KEY_LAST_TAB, TabIndex.SPEED_DIAL);
+            .getInt(KEY_LAST_TAB, TabIndex.CALL_LOG);

-        // If the voicemail tab cannot be shown, default to showing speed dial
+        // If the voicemail tab cannot be shown, default to showing call log
        if (tabIndex == TabIndex.VOICEMAIL && !canShowVoicemailTab) {
-            tabIndex = TabIndex.SPEED_DIAL;
+            tabIndex = TabIndex.CALL_LOG;
        }
    }
}
```

2.5.3 编译模块

编译模块: Dialer。具体的编译方法参见“1.2 编译方法”。

2.5.4 验证方式

开机之后, 打开拨号盘应用, 查看界面是否为显示通话记录的 Tab 界面。