

基于安卓系统的 车载蓝牙应用 分析设计文档

班级：计 34

组名：APA

组员：何晓楠 2013011361

姚炫容 2013011379

黄予 2013011363

杨晓成 2013011383

郑玉昆 2011011384

目录

1 车载蓝牙系统需求分析.....	4
1.1 项目背景	4
1.2 项目目标	4
1.3 运行环境	4
1.4 功能需求	4
1.5 性能需求	5
1.6 可用性需求.....	5
1.7 出错处理需求	5
2 Android 系统中的蓝牙架构分析	5
2.1 应用框架层 Application framework.....	6
2.2 蓝牙系统服务层 Bluetooth system service	6
2.3 JNI	6
2.4 硬件抽象层 HAL.....	6
2.5 蓝牙协议栈 Bluetooth Stack	6
3 HFP 协议分析	7
4 HFP 协议移植	8
4.1 版本说明	8
4.2 逐层移植代码	9
5 A2DP 协议分析.....	15
5.1 A2DP 调用分析.....	16
6 A2DP 协议移植.....	17

6.1 应用框架层	17
6.2 蓝牙系统服务层	18
6.3 蓝牙协议栈	18
7 PBAP 协议分析	19
8 PBAP 协议 移植及通讯录的实现	23
9 关于移植的其他问题	24
10 编译	24
11 蓝牙 APP 前端	27
11.1 界面设计原则	27
11.2 各界面设计详述	27

1 车载蓝牙系统需求分析

1.1 项目背景

近年来，归功于半导体芯片技术的大发展、大数据以及车联网等技术的出现与成熟，汽车的“大脑”——车载控制系统正在发生深刻的变革，汽车的主要创新从引擎盖下方扩展到仪表盘后方。在过去的几年中，汽车产业有超过 90%的创新都与智能化系统相关。

1.2 项目目标

本项目是基于 Android 的智能车机系统的一部分，拟设计和开发基于 Android 的车载蓝牙客户端，部署在车机系统上，通过蓝牙协议连接智能手机，实现电话接打、通讯录同步、音频传输等功能。

1.3 运行环境

Android 4.2.2 智能车载系统

1.4 功能需求

（1）蓝牙电话。在车机系统上实现支持蓝牙 HFP 协议的客户端，当车机系统与手机进行蓝牙配对后，可以从车机系统进行手机上电话的呼叫，来电的接听、拒接，通话的挂断等操作。

（2）蓝牙通讯录同步。在车机系统上实现支持蓝牙 PBAP 协议的客户端，当车机系统与手机进行蓝牙配对后，可以将手机上的通讯录同步到车机系统，可以从车机系统进行手机上通讯录的新建、编辑、删除、检索条目等操作。

（3）蓝牙音频传输。在车机系统上实现支持蓝牙 A2DP 协议的客户端，当车机系统与手机进行蓝牙配对后，可以从车机系统进行手机上音频播放、暂停、上一曲、下一曲等操作，可以以流的方式从手机上将音频同步传输至车机系统上播放（注意，必须以流的方式同步播放，不能直接将音频文件本身传输到车机系统上然后在车机系统上播放）。

（4）在车机系统上设计并实现 Android 应用程序，为上述功能提供完整的界面支持。

1.5 性能需求

(1) 数据精确度：进行诸如查找、删除、修改等操作请求时，由于其必须保证输入数据要与数据库数据相比配的原则，所以系统应保证响应数据的正确率，以及覆盖率。

(2) 时间特性：为满足用户的使用要求，数据的响应时间，更新时间，处理时间运行时间 都应控制在 1~2 秒之内。

1.6 可用性需求

(1) 界面简洁易懂，用户学习成本低；

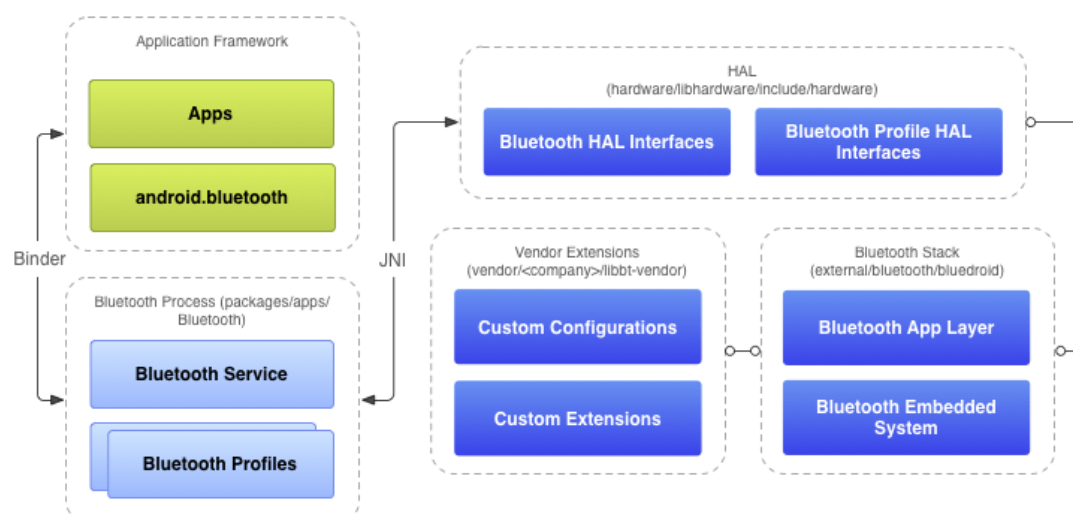
(2) 操作完成时有统一规范的提示信息，以防止用户误操作；

(3) 在任何时候主机或备份机上的软件应该至少有一个是可以正常使用的，且在一周之内任何一台计算机上该软件不可用的时间不能超过总时间的 10%。

1.7 出错处理需求

如果发生了环境错误，软件应能够首先自行进行出错处理，例如查找哪个 部分出错并且上报给管理员。如果软件错误响应了错误环境信息并导致大的崩溃或是长时间无法响应操作要求时，应能在用户可以接受的范围之内将软件关闭。

2 Android 系统中的蓝牙架构分析



2.1 应用框架层 Application framework

在应用框架层代码最上层为 Application Framework，蓝牙相关代码位于 `android/frameworks/base/core/java/android/bluetooth` 下，代码语言为 java，该层的代码会编译进 android sdk，从而将蓝牙相关的 API 通过 sdk 暴露给应用程序，应用程序通过这一层的 API 来调用蓝牙功能。该层通过 Binder 机制来调用第二层的代码，具体的实现方式为通过 AIDL（Android Interface Definition Language）进行跨进程通信。可以看到，文件夹下存在 .java 文件与 .aidl 文件，其中，.java 文件中的代码会调用 .aidl 中声明的接口，.aidl 文件中的代码只声明了接口，未定义实现，实现位于第二层代码。通过这样的方式，层与层充分地解耦和，下层向上层提供服务，只要定义好接口，下层的实现可以随时改变而不影响上层。

。

2.2 蓝牙系统服务层 Bluetooth system service

蓝牙系统服务位于 `packages/apps/Bluetooth`，被作为一个 Android 的系统 App。它在 Android 框架层实现了蓝牙的 Service 和 Profile，相关代码位于 `android/packages/apps/Bluetooth` 下，

代码语言为 java，该层实现了上层声明的接口，蓝牙状态机的实现也位于该层。该层通过 JNI（Java Native Interface）方式调用 HAL 层的代码，JNI 是 java 平台的一部分，它使得 java 代码能够与其他语言写的代码进行交互，由于协议栈第三层使用 c++ 语言实现，故需要通过 JNI 调用 HAL 层的 c++ 代码。

2.3 JNI

与 `android.bluetooth` 对应的 JNI 代码位于 `packages/apps/Bluetooth/jni`，JNI 层代码调用到 HAL 层，并当发生某些蓝牙动作的时候，接受来自 HAL 层的回调。

2.4 硬件抽象层 HAL

硬件抽象层（Hardware Abstraction Layer）定义了 `android.bluetooth` API 和蓝牙进程需要用到的标准接口。相关代码位于 `android/hardware/libhardware/include/hardware`，代码语言为 c++。可以看到，该层均为 .h 头文件，负责声明一些公共的结构、函数等以供蓝牙协议栈使用。

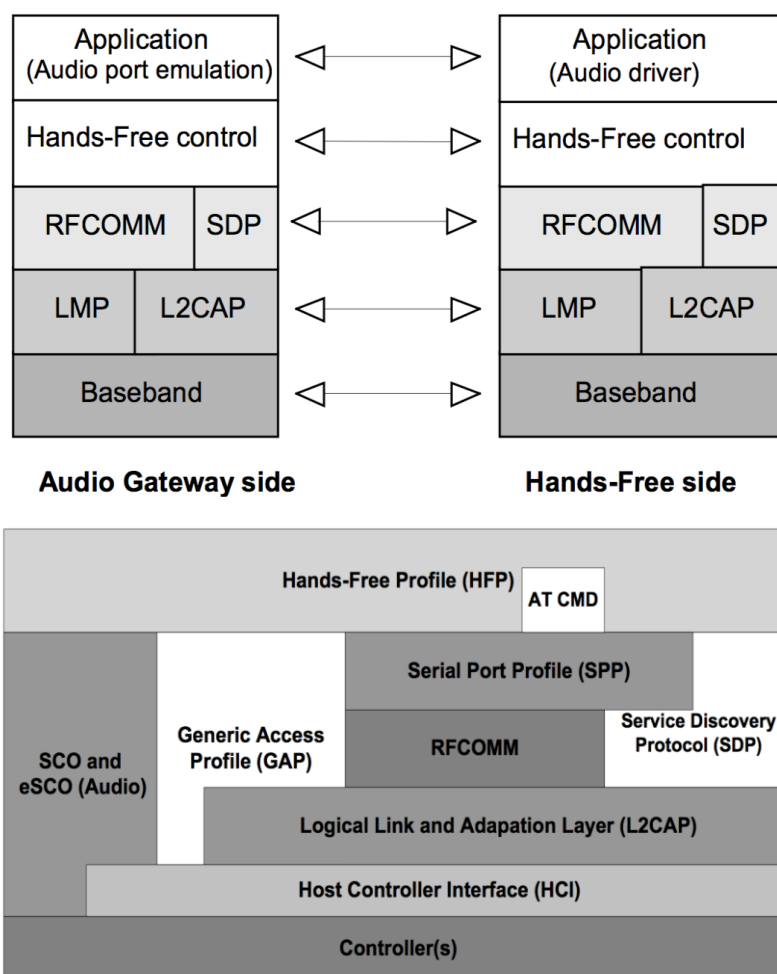
2.5 蓝牙协议栈 Bluetooth Stack

Android 提供的默认蓝牙协议栈位于 `android/external/bluetooth/bluedroid`。这个协议栈

实现了通用的蓝牙 HAL 接口，代码语言为 c++，作为最底层，是整个蓝牙模块的核心部分，实现了蓝牙设备管理、音频传输等功能。

3 HFP 协议分析

HFP(Hand-Free Profile)免提协议提供了让蓝牙设备控制电话的接听、挂断、拒接、重拨等功能。HFP 定义了两种角色，音频网关 AG (Audio Gateway) 与免提设备 HF (Hands-Free)，协议栈如下图所示：



AG 指控制音频输入输出的网关设备，典型设备如手机；HF 指可收发音频、遥控 AG 的免提设备，典型设备如蓝牙耳机。

AG 与 HF 建立连接的过程分两个阶段：建立服务连接与建立音频连接。建立服务连接时，AG 与 HF 基于 ACL 链路建立 RFCOMM 连接，通过 AT 命令完成双方基本信息的交互。服务连接建立完毕后，当 AG 端检测到通话时，将与 HF 端建立音频连接，通过 SCO 链路进行音频传输。

HFP 协议提供的主要功能如下图：

	Feature	Support in HF	Support in AG
1.	Connection management	M	M
2.	Phone status information	M ^(note 1)	M
3.	Audio Connection handling	M	M
4.	Accept an incoming voice call	M	M
5.	Reject an incoming voice call	M	O
6.	Terminate a call	M	M
7.	Audio Connection transfer during an ongoing call	M	M
8.	Place a call with a phone number supplied by the HF	O	M
9.	Place a call using memory dialing	O	M
10.	Place a call to the last number dialed	O	M
11.	Call waiting notification	O	M
12.	Three-way calling	O ^(note 2)	O ^(note 3)
13.	Calling Line Identification (CLI)	O	M
14.	Echo canceling (EC) and noise reduction (NR)	O	O
15.	Voice recognition activation	O	O
16.	Attach a Phone number to a voice tag	O	O
17.	Ability to transmit DTMF codes	O	M
18.	Remote audio volume control	O	O
19.	Respond and Hold	O	O
20.	Subscriber Number Information	O	M
21a.	Enhanced Call Status	O	M
21b.	Enhanced Call Controls	O	O
22.	Individual Indicator Activation	O	M

	Feature	Support in HF	Support in AG
23	Wide Band Speech	O	O
24	Codec Negotiation	O ^(note 4)	O ^(note 4)
25	HF Indicators	O	O

Note 1: The HF shall support at least the two indicators "service" and "call".

Note 2: If "Three-way calling" is supported by the HF, it shall support AT+CHLD values 1 and 2. The HF may additionally support AT+CHLD values 0, 3 and 4.

Note 3: If "Three-way calling" is supported by the AG, it shall support AT+CHLD values 1 and 2. The AG may additionally support AT+CHLD values 0, 3, and 4.

Note 4: If Wide Band Speech is supported, Codec Negotiation shall also be supported.

其中提供的主要功能如下：

1. 接听来电
2. 拒接来电
3. 挂断通话
4. 拨号
5. 音量控制

4 HFP 协议移植

4.1 版本说明

原版本：Android 4.2.2.

新版本：Android 5.0.1.r1

4.2 逐层移植代码

1. Application Framework

- (1) android/frameworks/base/core/java/android/Bluetooth
 - 1) 直接拷贝文件
IBluetoothHeadsetClient.adil
IBluetoothHeadsetClient.java
IBluetoothHeadsetClientCall.aidl
IBluetoothHeadsetClientCall.java
- (2) android/frameworks/base/Android.mk
 - 1) 增加代码，使得（1）加入的文件得到编译
core/java/android/bluetooth/IBluetoothHeadsetClient.aidl \
- (3) android/frameworks/base/core/java/android/content/Intent.java
 - 1) 引入头文件
import java.util.List
import android.content.pm.ApplicationInfo;
 - 2) 增加函数
public ComponentName resolveSystemService(PackageManager pm, int flags)
- (4) android/frameworks/base/core/java/android/content/Context.java
 - 1) 引入头文件
import android.annotation.SystemApi;
 - 2) 增加函数
public boolean bindServiceAsUser(Intent service, ServiceConnection conn, int flags, UserHandle user)
- (5) android/frameworks/base/core/java/android/bluetooth/BluetoothProfile.java
 - 1) 增加常量
static public final int GATT
static public final int GATT_SERVER
public static final int MAP
public static final int A2DP_SINK
public static final int AVRCP_CONTROLLER
public static final int HEADSET_CLIENT
- (6) android/frameworks/base/core/java/android/annotation
 - 1) 直接拷贝文件
SystemApi.java
- (7) android/frameworks/base/core/java/android/bluetooth/BluetoothAdapter.java
 - 1) getProfileProxy 函数中增加分支

- if (profile == BluetoothProfile.HEADSET_CLIENT)
 - 2) closeProfileProxy 函数中增加分支
 - case BluetoothProfile.HEADSET_CLIENT:
 - (8) android/frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/policy
 - 1) 直接拷贝文件
 - BluetoothUtil.java
 - (9) android/frameworks/base/core/java/android/bluetooth/BluetoothUuid.java
 - 1) 增加常量
 - public static final ParcelUuid HOGP
 - public static final ParcelUuid MAP
 - public static final ParcelUuid MNS
 - public static final ParcelUuid MAS
 - public static final ParcelUuid BASE_UUID
 - public static final int UUID_BYTES_16_BIT
 - public static final int UUID_BYTES_32_BIT
 - public static final int UUID_BYTES_128_BIT
 - (10) android/frameworks/base/core/java/android/content/ContextWrapper.java
 - 1) 增加函数
 - public boolean bindServiceAsUser(Intent service, ServiceConnection conn, int flags, UserHandle user)
 - (11) android/frameworks/base/core/java/android/app/ContextImpl.java
 - 1) 增加函数
 - public boolean bindServiceAsUser(Intent service, ServiceConnection conn, int flags, UserHandle user)
 - 2) 增加函数
 - private boolean bindServiceCommon(Intent service, ServiceConnection conn, int flags, UserHandle user)
 - 3) 增加函数
 - private void validateServiceIntent(Intent service)
 - (12) android/frameworks/base/core/java/android/os/StrictMode.java
 - 1) 增加函数
 - public static void onFileUriExposed(String location)
 - 2) 增加函数
 - public static boolean vmFileUriExposureEnabled()
 - 3) 增加常量
 - private static final int DETECT_VM_FILE_URI_EXPOSURE
 - (13) android/frameworks/base/core/java/android/content/Intent.java
 - 1) 增加函数

- ```
public void prepareToLeaveProcess()
```
- 2) 引入头文件
- ```
import android.os.StrictMode;
```
- (14) android/frameworks/base/core/java/android/content/ClipData.java
- 1) 增加函数
- ```
public void prepareToLeaveProcess()
```
- 2) 引入头文件
- ```
import android.os.StrictMode;
```
- (15) android/frameworks/base/core/java/android/net/Uri.java
- 1) 增加函数
- ```
public void checkFileUriExposed(String location)
```
- 2) 引入头文件
- ```
import android.os.StrictMode;
```
- (16) android/frameworks/base/core/java/android/os/Build.java
- 1) 增加常量
- ```
public static final int L
public static final int LOLLIPOP
```

## 2. Bluetooth Process

- (1) android/packages/apps/Bluetooth/src/com/android/Bluetooth
- 1) 直接拷贝文件夹
- ```
hfpclient
```
- (2) android/packages/apps/Bluetooth/jni/
- 1) 直接拷贝文件
- ```
com_android_bluetooth_hfpclient.cpp
```
- (3) android/packages/apps/Bluetooth/jni/Android.mk
- 1) 增加代码，使得使得 (2) 加入的文件得到编译
- ```
com_android_bluetooth_hfpclient.cpp \
```
- (4) android/packages/apps/Bluetooth/res/values/config.xml
- 1) 增加代码，打开 hfpclient 的开关
- ```
<bool name="profile_supported_hfpclient">true</bool>
```
- (5) android/packages/apps/Bluetooth/src/com/android/bluetooth/btservice/Config.java
- 1) 引入头文件
- ```
import com.android.bluetooth.hfpclient.HeadsetClientService;
```
- 2) 在 PROFILE_SERVICES 数组中加入新项
- ```
HeadsetClientService.class
```
- 3) 在 PROFILE\_SERVICES\_FLAG 数组中加入新项
- ```
R.bool.profile_supported_hfpclient
```

(6) android/packages/apps/Bluetooth/AndroidManifest.xml

1) 增加服务

```
<service
    android:process="@string/process"
    android:name=".hfpclient.HeadsetClientService"
    android:enabled="@bool/profile_supported_hfpclient">
    <intent-filter>
        <action android:name="android.bluetooth.IBluetoothHeadsetClient" />
    </intent-filter>
</service>
```

(7) android/packages/apps/Bluetooth/jni/ com_android_bluetooth.h

1) 增加函数声明

```
int register_com_android_bluetooth_hfpclient(JNIEnv* env);
```

(8) android/packages/apps/Bluetooth/jni/com_android_bluetooth_btservice_AdapterService.cpp

1) 在函数 JNI_OnLoad 中增加分支

```
if ((status = android::register_com_android_bluetooth_hfpclient(e)) < 0)
```

3. HAL

(1) android/hardware/libhardware/include/hardware

1) 直接拷贝文件

```
bt_hf_client.h
```

(2) android/hardware/libhardware/include/hardware/bluetooth.h

1) 引入头文件

```
#include <stdbool.h>
```

2) 增加宏定义

```
#define BT_PROFILE_HANDSFREE_CLIENT_ID "handsfree_client"
```

3) 增加代码

```
typedef void (*alarm_cb)(void *data);
```

4) 增加代码

```
typedef bool (*set_wake_alarm_callout)(uint64_t delay_millis, bool
should_wake, alarm_cb cb, void *data);
```

5) 增加代码

```
typedef int (*acquire_wake_lock_callout)(const char *lock_name);
```

6) 增加代码

```
typedef int (*release_wake_lock_callout)(const char *lock_name);
```

7) 增加结构体

```
bt_os_callouts_t
```

4. Bluedroid

(1) android/external/bluetooth/bluedroid/btif/src

1) 直接拷贝文件

```
btif_hf_client.c
```

- (2) android/external/bluetooth/bluedroid/bta/
 - 1) 直接拷贝文件夹
 - hf_client
- (3) android/external/bluetooth/bluedroid/bta/include
 - 1) 直接拷贝文件
 - bta_hf_client_api.h
- (4) android/external/bluetooth/bluedroid/bta/Android.mk
 - 1) 增加代码，使得 (2) 加入的文件得到编译
 - ./hf_client/bta_hf_client_act.c \
 - ./hf_client/bta_hf_client_api.c \
 - ./hf_client/bta_hf_client_at.c \
 - ./hf_client/bta_hf_client_cmd.c \
 - ./hf_client/bta_hf_client_main.c \
 - ./hf_client/bta_hf_client_rfc.c \
 - ./hf_client/bta_hf_client_sco.c \
 - ./hf_client/bta_hf_client_sdp.c \

 - \$(LOCAL_PATH)/hf_client \
 - \$(LOCAL_PATH)/../utils/include \
- (5) android/external/bluetooth/bluedroid/main/Android.mk
- (6) android/external/bluetooth/bluedroid/utils/include/bt_utils.h
 - 1) 增加宏定义
 - #define UNUSED(x) (void)(x)
- (7) android/external/bluetooth/bluedroid/include/bt_trace.h
 - 1) 增加代码
 - #define BT_TRACE(l,t,...)至
 - #define bdl(assert_if) ((void)0)
 - #endif
 - 之间的部分
- (8) android/external/bluetooth/bluedroid/btif/include/btif_common.h
 - 1) 增加宏定义
 - #define BTIF_HF_CLIENT 5
 - 2) 增加代码
 - BTIF_HF_CLIENT_CLIENT_CB_START = BTIF_SIG_CB_START(BTIF_HF_CLIENT),
 - BTIF_HF_CLIENT_CB_AUDIO_CONNECTING, /* AUDIO connect has been sent to BTA successfully */
- (9) android/external/bluetooth/bluedroid/btif/src/btif_hf_client.c
 - 1) 替换代码
 - 将 static bt_status_t connect_int(bt_bdaddr_t *bd_addr, uint16_t uuid)
 - 替换为 static bt_status_t connect_int(bt_bdaddr_t *bd_addr)

- (10) android/external/bluetooth/bluedroid
 - 1) 创建文件夹
 - osi
 - osi/test
 - osi/src
 - osi/include
- (11) android/external/bluetooth/bluedroid/osi/test/
 - 1) 直接拷贝文件
 - list_test.cpp
- (12) android/external/bluetooth/bluedroid/osi/src/
 - 1) 直接拷贝文件
 - list.c
- (13) android/external/bluetooth/bluedroid/osi/include/
 - 1) 直接拷贝文件
 - list.h
 - osi.h
- (14) android/external/bluetooth/bluedroid/osi
 - 1) 增加 makefile 文件
 - Android.mk, 其中内容见原文件
- (15) android/external/bluetooth/bluedroid/main/Android.mk
 - 1) 增加代码, 使得 (1) 加入的文件得到编译
 - ../btif/src/btif_hf_client.c \
 - 1) 增加代码, 使得 (10) 增加的文件夹下的文件得到编译
 - \$(LOCAL_PATH)/../osi/include \
 - 2) 增加代码, 对 utils 以及 osi 进行静态编译
 - LOCAL_STATIC_LIBRARIES += \
 - libbt-utils \
 - libosi \
- (16) android/external/bluetooth/bluedroid/utils/Android.mk
 - 1) 替换代码
 - 将 LOCAL_MODULE_CLASS := SHARED_LIBRARIES 替换为
 - LOCAL_MODULE_CLASS := STATIC_LIBRARIES
- (17) android/external/bluetooth/bluedroid/btif/include/btif_util.h
 - 1) 增加函数声明
 - const char* dump_hf_client_event(UINT16 event);
- (18) android/external/bluetooth/bluedroid/btif/src/btif_util.c
 - 1) 增加函数
 - const char* dump_hf_client_event(UINT16 event)
 - 2) 引入头文件

```
#include "bta_hf_client_api.h"
```

(19) android/external/bluetooth/bluedroid/btif/src/bluetooth.c

1) 引入头文件

```
#include <hardware/bt_hf_client.h>
```

2) 增加代码

```
extern bthf_client_interface_t *btif_hf_client_get_interface();
```

3) 在函数 get_profile_interface 中增加分支

```
if (is_profile(profile_id, BT_PROFILE_HANDSFREE_CLIENT_ID))  
    return btif_hf_client_get_interface();
```

(20) android/external/bluetooth/bluedroid/btif/src/btif_dm.c

1) 增加函数声明

```
extern bt_status_t btif_hf_client_execute_service(BOOLEAN b_enable);
```

2) 在函数 btif_in_execute_service_request 中增加分支

```
case BTA_HFP_HS_SERVICE_ID:
```

5. droidoc

由于编译 droiddoc 需要较大的内存空间, 当为 linux 虚拟机分配的内存不足时, 编译 droiddoc 时将导致编译终止。droidoc 仅仅与自动生成的 android 源码说明文档有关, 对 Android 系统的运行没有影响, 所以可删去。

(1) android/build/core/droiddoc.mk

1) 删除代码, 使得 droiddoc 不参与编译。

```
ALL_DOCS += $(full_target)
```

(2) android/frameworks/base/Android.mk

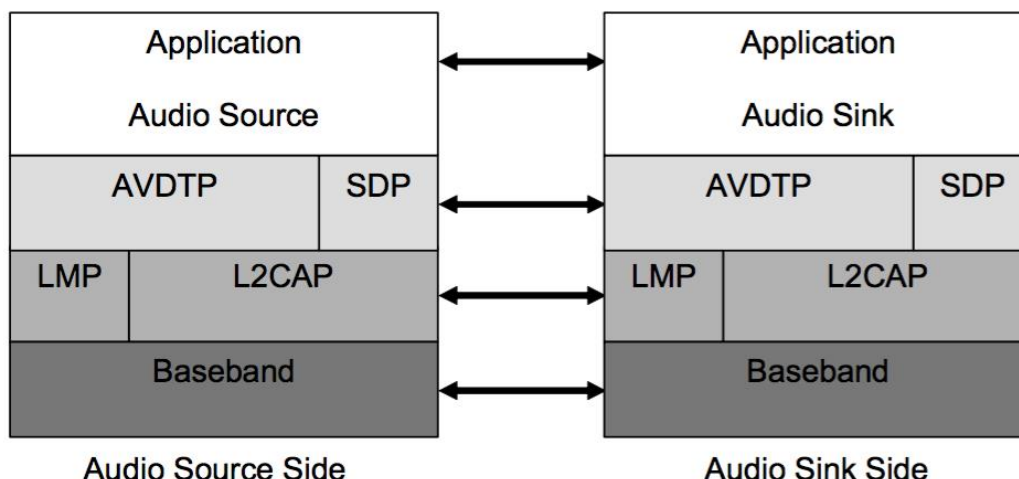
1) 删除代码

```
include $(CLEAR_VARS)至
```

```
droidcore: doc-comment-check-docs 之间的所有代码
```

5 A2DP 协议分析

A2DP 的协议栈如下图所示, 具体用到了 avdtp 和 l2cap 的相关部分。



5.1 A2DP 调用分析

首先，Bluetooth 作为一个系统应用，在安卓源代码中实现于 `android/platform/packages/apps/Bluetooth/` 下。我们需要在其中打开对于 A2dp 客户端的支持。具体改动如下：

1). `android/platform/packages/apps/Bluetooth/AndroidManifest.xml`

增加内容 `<uses-permission android:name="android.permission.RECORD_AUDIO" />`。给予该应用音频纪录的权限。

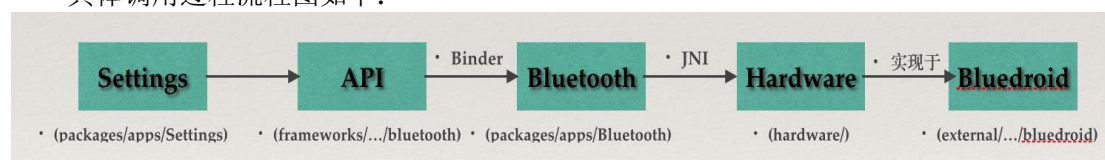
2). `android/platform/packages/apps/Bluetooth/res/values/config.xml`

增加内容 `<bool name="profile_supported_a2dp_sink">true</bool>`

根据 A2DP 协议的规定，音频数据的来源称为 source，接收音频数据的车载蓝牙系统称为 sink。增加该内容后，将用户的手机与车载蓝牙系统以蓝牙相连，在用户手机的设置中会出现“媒体音频”一项。

选中该项后，若车载蓝牙系统 A2dp client 端的具体功能已近实现，在用户手机播放音乐时，音频数据将不再发送到用户手机的扬声器，转而经由发送到车载蓝牙系统中，再由车载蓝牙系统的扬声器播放，这样就实现了 A2dp 的音频传输功能。

具体调用过程流程图如下：



1. 由系统设置（`android/platform/packages/apps/Settings/`）中的与蓝牙有关各选项调用 Application Frameworks（`android/platform/frameworks/base/core/java/android/bluetooth/`）中的 API。
2. 由 Application Frameworks 通过 Binder IPC 机制调用系统应用 Bluetooth 进程（`android/platform/packages/apps/Bluetooth/`）。
3. 由 Bluetooth 进程通过 JNI 调用访问硬件抽象层 HAL（`android/platform/hardware/`）。
4. 硬件功能的具体实现及协议栈的搭建（`android/platform/external/bluetooth/bluedroid/`）。

6 A2DP 协议移植

A2dp 的移植主要参考 <https://android-review.googlesource.com/#/c/98161/> 和 android5.0.0 r1 的源代码。

6.1 应用框架层

在这一层，主要修改是把 A2DP profile 作为一个音频输入设备。其余的修改都是一些围绕这一修改进行的。

(1) `frameworks/base/media/java/android/media/`

1. `AudioManager.java`
2. `AudioService.java`
3. `AudioSystem.java`
4. `MediaRecorder.java`

(2) `hardware/libhardware_legacy/audio/`

1. `Android.mk`
2. `AudioPolicyManagerBase.cpp`

(3) `system/core/include/system/`

1. `audio.h`

(4) `device/softwinner/common/hardware/audio/`

1. `audio_policy.conf`

6.2 蓝牙系统服务层

在这一层中，主要是对 a2dp 状态机的相关代码进行修改。本质是把 source 的角色转换为 sink。代码中主要增加的部分是：把 a2dp 得到的音频流数据用 AudioRecord 记录下来，然后用 AudioTrack 播放出来。

- (1) packages/apps/Bluetooth/src/com/android/Bluetooth/a2dp/
 - 1. A2dpStateMachine.java

6.3 蓝牙协议栈

蓝牙协议栈这部分较为复杂，移植的基本思路是把 a2dp 用到的相关函数从 5.0 直接复制过来，同时把相关的 hal 接口作简单修改。

- (1) external/bluetooth/bluedroid
 - 1. Android.mk
- (2) external/bluetooth/bluedroid/audio_a2dp_hw/
 - 1. Android.mk
 - 2. audio_a2dp_hw.h
 - 3. audio_a2dp_hw.c
- (3) external/bluetooth/bluedroid/bta/av/
 - 1. bta_av_aact.c
 - 2. bta_av_act.c
 - 3. bta_ac_int.h
 - 4. bta_av_sbc.c
 - 5. bta_ac_ssm.c
- (4) external/bluetooth/bluedroid/bta/include/
 - 1. bta_av_co.h
 - 2. bta_ac_sbc.h
- (5) external/bluetooth/bluedroid/btif/co/
 - 1. bta_av_co.c
- (6) external/bluetooth/bluedroid/btif/include
 - 1. btif_av_api.h
 - 2. btif_media.h

(7) external/bluetooth/bluedroid/btif/src

1. btif_av.c
2. btif_dm.c
3. btif_media_task.c
4. btif_storage.c

(8) external/bluetooth/bluedroid/main

1. Android.mk

(9) external/bluetooth/bluedroid/stack/avdt/

1. avdt_scb_act.c

(10)external/udrv/ulinux/

1. uipc.c

A2dp 默认需要支持 SBC 的编码，sbc 的编码解码部分也是从 5.0 直接复制过来。

(11)external/embdrv/sbc/

7 PBAP 协议分析

根据 PBAP 协议，其对于底层协议的依赖图如图 2. 1-1。

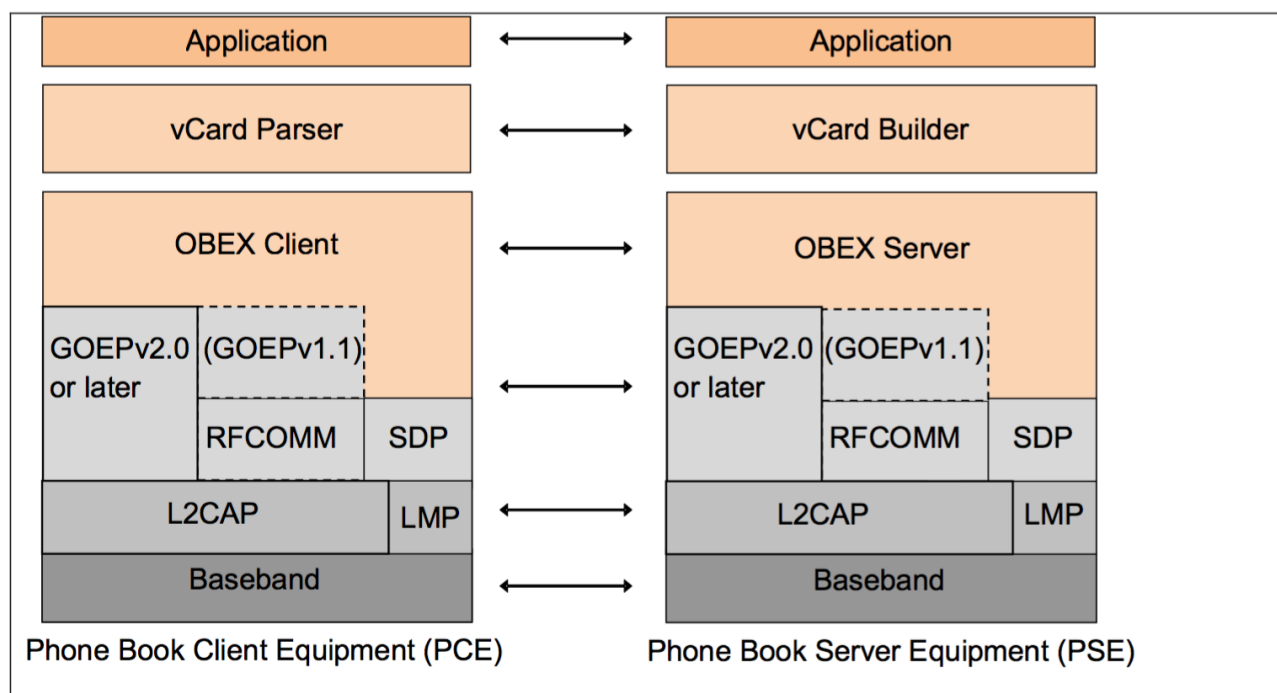


图 2.1-1 PBAP 协议栈

定义车载蓝牙系统 cubietruck 为通讯录客户端 (Phone Book Client Equipment, PCE), 与其通过蓝牙相连接的手机定义为通讯录服务器端 (Phone Book Server Equipment, PSE)。功能表现为, 当车载蓝牙系统 PCE 通过蓝牙连接到手机 PSE 后, PCE 向 PSE 发送通讯录访问请求, 随后手机 PSE 端弹出是否允许车载蓝牙系统访问手机通讯录的提示框, 用户点击同意后, 车载蓝牙系统 PCE 端应该访问到 PSE 端的通讯录。

图中的 vCard 部分即我们需要移植的部分。该部分向下基于 OBEX 协议 (Object Exchange profile), 向上为我们设计的 APK 提供能够访问个人手机通讯录的接口。在 Cubietruck 基于的安卓 4.2.2 系统中, OBEX 协议已经实现, 且可以通过 SDK 调用其接口。故 PBAP 协议的移植无需涉及安卓底层系统的更改, 可以完全在 APK 所在的应用层实现。

PBAP 协议共定义了 4 个对外接口, 为了实现所要求的功能, 至少需要调用其中两个, 列举如下:

1. PullPhoneBook 下载具体的通讯录内容, 其流程图如 2.2.1-1 所示。

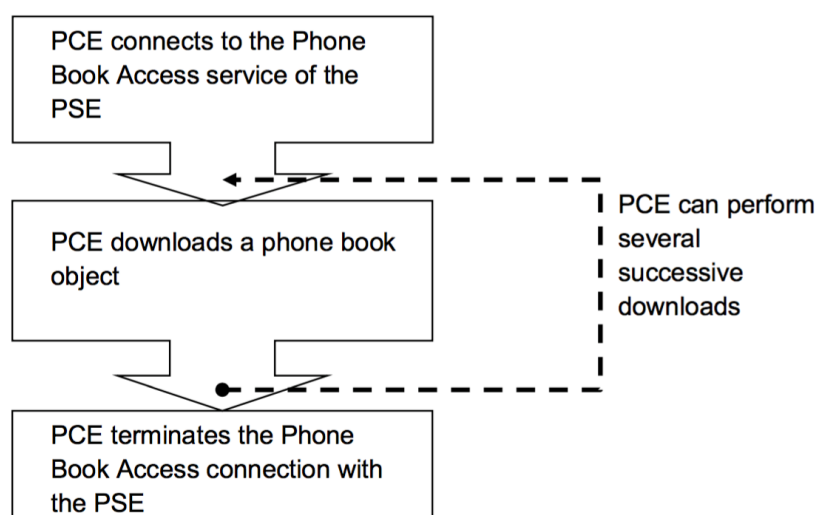


图 2.2.1-1 PullPhoneBook 序列图

所下载的通讯录序列依升序排列，该函数不进行排序。

2. PullvCardListing

同样是获取当前路径下的所有联系人信息。与 PullPhoneBook 函数不同的是，此处获得的序列在函数内部可以进行修改和排序。具体原理详见函数“3 SetPhoneBook”。

3. SetPhoneBook 指令将要下载的通讯录序列在手机 PSE 端中的存储路径。

根据 PBAP 协议说明，通讯录在手机中的存储位置有两种可能，一是存储在手机本地，二是存储到 SIM 卡上。

同时根据 PBAP 协议，手机内部将通讯录对象分为 7 种类型，依次列举如下：

- (1) 主要通讯录对象 (Main Phone Book object, pb)，包含通讯录中所有联系人信息。
- (2) 接入历史对象 (Incoming Calls History object, ich)，包含所有来电信息。
- (3) 拨出历史对象 (Outgoing Calls History object, och)，包含所有拨号信息。
- (4) 未接来电历史对象 (Missed Calls History object, msh)
- (5) 最近联系对象 (Combined Calls History object, cch)，包含 (2) (3) (4) 中中所有内容
- (6) 快速拨号对象 (Speed-Dial object, spd)
- (7) 重要联系人对象 (Favorite Contacts object, fch)

具体的存储路径如图 2.2.1-2 所示。

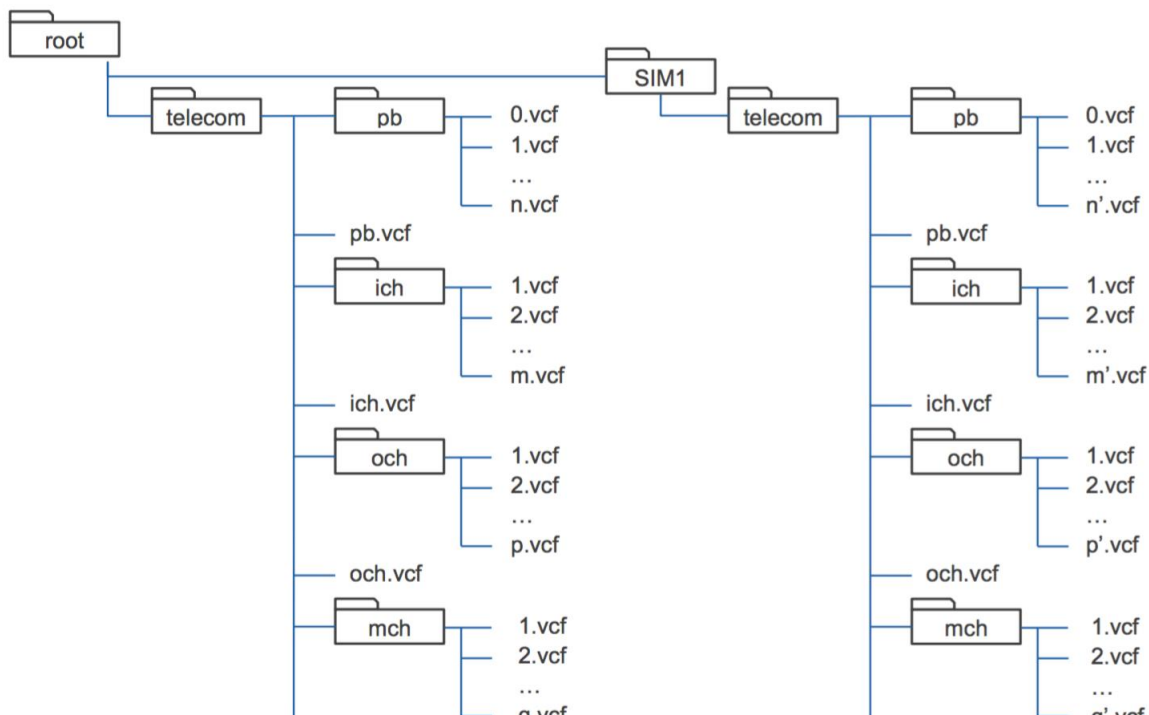


图 2. 2. 1-2 PBAP 虚拟文件结构

以主要通讯录对象 pb 为例。pb.vcf 所含内容与 pb/文件夹下所有<handle.vcf>文件所包含的内容之和是一致的。不同之处在于，使用 PullPhoneBook 函数获得的是 pb.vcf 中的内容；而使用 PullvCardListing 函数获取的是 pb/文件夹下的内容，由此后者可以进行修改和排序而前者不能。

SetPhoneBook 与 PullvCardListing 协同工作的序列图如图 2. 2. 1-3。

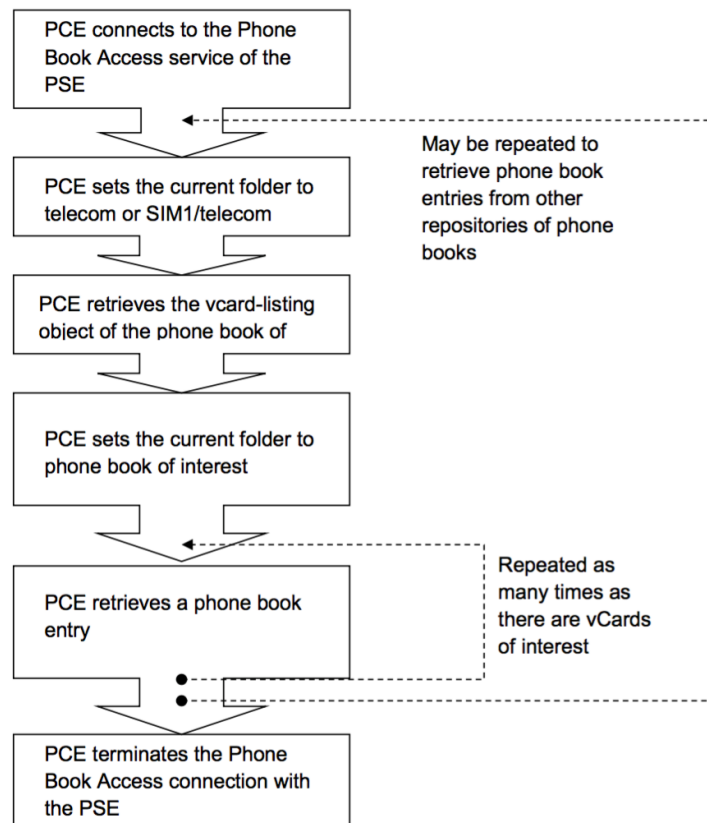


图 2. 2. 1-3 通讯录浏览序列图

8 PBAP 协议 移植及通讯录的实现

我们在安卓 5.0.1_r1 版本（源代码来源 <https://android.googlesource.com>）中的文件路径 `android/platform/frameworks/opt/bluetooth/src/android/bluetooth/client/pbap` 下找到了 PBAP 协议的客户端实现。

为了能够调用 PBAP 协议提供的一系列函数，我们把这些代码放在了 APK 源码的文件路径 `/src/com/bluetooth/android/bluetooth/client/pbap` 下。

同时安卓 5.0.1_r1 下 PBAP 协议调用 OBEX 协议的接口是通过 `javax` 包，文件路径为 `android/platform/frameworks/base/obex/javax/obex/`，我们将其移植到 APK 源码中的文件路径 `/src/com/javax/obex/` 下。

PBAP 协议中传输时的数据格式 `vcard` 的定义与解析文件位于安卓 5.0.1_r1 文件路径 `android/platform/frameworks/opt/vcard` 下，我们将其移植到 APK 源码中的文件路径 `/src/com/bluetooth/android/vcard/` 下。

开始 PBAP 连接前，我们需要一个已经在车载蓝牙系统 PCE 和用户手机 PSE 间建立了蓝牙连接的蓝牙设备对象 `BluetoothDevice`，它由 APK 的其它部分提供。随后，新建一个 `BluetoothPbapClient` 对象，这个对象在 `/src/com/bluetooth/android/bluetooth/client/pbap` 中定义，将其与 `BluetoothDevice` 和一个新的 `Handle` 进行绑定。

这个新的 `Handle` 将用于获取的通讯录信息的处理。为此，我们需要重写函数

```
public void handleMessage(Message msg)
```

我们使用的 PBAP 接口为 `SetPhoneBook+PullPhoneBook` 类型。

`msg.obj` 的内容为 `ArrayList<VCardEntry>`。对其中的每一个联系人，姓名和电话信息分别存储在 `VCardEntry.NameData` 和 `VCardEntry.PhoneData` 下。与 `vCard` 有关内容的定义位于 `android/platform/frameworks/opt/vcard` 下。

具体使用的 PBAP 接口接口如下。

```
bool setPhoneBookFolderRoot()
```

将当前目录调整至系统根目录。返回 `true` 表示操作成功，`false` 表示操作失败或超时。

```
bool pullPhoneBook(String)
```

此处进行的操作为 `SetPhoneBook` 和 `PullPhoneBook`。`String` 中为具体的路径。这里我们使用了两个文件路径，分别为 `“/telecom/pb.vcf”` 和 `“/SIM1/telecom/pb.vcf”`，分别代表手机和 SIM 卡中通讯录内容。获得的通讯录信息在 `Handle` 中的 `handleMessage` 函数中进行处理。

9 关于移植的其他问题

1. 关于@hide 注释

可以看到，Android 5.0 中与 hfp client 相关的代码均带有@hide 注释，该注释会使得在编译 Android sdk 时相应的接口在 sdk 中隐藏，从而无法被上层应用程序调用。android/out/target/common/obj/JAVA_LIBRARIES/framework_intermediates 路径下的 classes.jar 文件则包含了所有的编译成果，包括隐藏的与未隐藏的。在开发上层应用程序时，只需将该.jar 包作为第三方库加入工程，并设置编译优先级高于 Android sdk，即可调用被@hide 标记的接口。由于工程第三方库有大小限制，为避免加入的.jar 包过于庞大，需要对其进行删减。在本项目中，仅留下.jar 包中的蓝牙相关部分即可。

2. 关于@link 注释

该注释仅与编译 droiddoc 有关，无须理会。

3. 关于 sdk

事实上 Android sdk 完全没有必要制作，只要采用 1 中所述的提取.jar 包作为第三方库的方法，使用 Android 4.2 的官方 sdk，应用程序便可通过编译并正确运行。

10 编译

1. 初次编译

(1) 配置编译环境

1) 系统：Ubuntu 12.04

2) 安装 JDK

A. 下载 JDK:

```
wget dl.cubieboard.org/software/tools/android/jdk1.6.0_45.tar.gz
```

B. 解压

```
sudo tar -xvf jdk1.6.0_45.tar.gz
```

C. 用编辑器打开.bashrc: sudo vim ~/.bashrc

增加三行:

```
JAVA_HOME=/jdk-path/jdk1.6.0_45
```

```
export JRE_HOME=/jdk-path/jdk1.6.0_45/jre
```

```
export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
```

注: jdk-path 为下载解压后的 jdk 路径

D. 验证是否成功:

```
source ~/.bashrc&&java -version
```



```
le@le:~$ source ~/.bashrc&&java -version
java version "1.6.0_45"
Java(TM) SE Runtime Environment (build 1.6.0_45-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed mode)
```

若如上图所示即代表安装成功

3) 安装编译所需包

依次运行如下命令：

- A. `sudo apt-get update`
- B. `sudo apt-get upgrade`
- C. `sudo apt-get install build-essential u-boot-tools uboot-mkimage
binutils-arm-linux-gnueabi`
- D. `sudo apt-get install gcc g++ gcc-arm-linux-gnueabi
gcc-arm-linux-gnueabi g++-multilib`
- E. `sudo apt-get install cpp-arm-linux-gnueabi libusb-1.0-0
libusb-1.0-0-dev wget fakeroot`
- F. `sudo apt-get install kernel-package zlib1g-dev libncurses5-dev
build-essential`
- G. `sudo apt-get install texinfo texlive ccache zlib1g-dev gawk bison flex
gettext uuid-dev`
- H. `sudo apt-get install ia32-libs git gnupg flex bison gperf build-essential zip`
- I. `sudo apt-get install curl libc6-dev x11proto-core-dev libx11-dev:i386
lib32ncurses5-dev`
- J. `sudo apt-get install libreadline6-dev:i386 mingw32 tofrodos
python-markdown`
- K. `sudo apt-get install libxml2-utils xsltproc zlib1g-dev:i386 libgl1-mesa-dev`

4) 下载源码包

解压后源码分为两个部分，android 与 lichee。

(2) 内核编译

A. 在 lichee/linux-3.4 目录下运行命令：

```
sudo cp arch/arm/configs/cubieboard2_config .config
```

B. 在 lichee 目录下运行命令：

```
./build.sh -p sun7i_android
```

C. 内核编译完成后，在 lichee/out 目录下可看到编译的产物

(3) 整体编译

A. 在 android 目录下依次运行命令：

```
source build/envsetup.sh
```

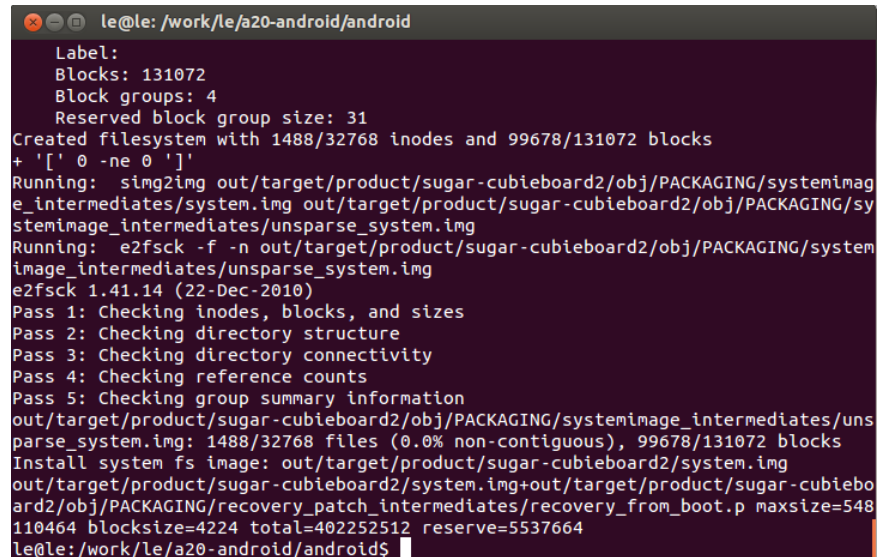
```
lunch
```

输入 16 （选择 sugar_cubietruck-eng）

extract-bsp

make -j8 （注：若 linux 系统只有 2G 内存，则 make -j4 或使用更少的线程编译，否则编译将由于内存不足而终止）

B. 当出现以下图示代表编译成功



```
le@le: /work/le/a20-android/android
Label:
Blocks: 131072
Block groups: 4
Reserved block group size: 31
Created filesystem with 1488/32768 inodes and 99678/131072 blocks
+ '[' 0 -ne 0 ']'
Running: simg2img out/target/product/sugar-cubieboard2/obj/PACKAGING/systemimage_intermediates/system.img out/target/product/sugar-cubieboard2/obj/PACKAGING/systemimage_intermediates/unsparse_system.img
Running: e2fsck -f -n out/target/product/sugar-cubieboard2/obj/PACKAGING/systemimage_intermediates/unsparse_system.img
e2fsck 1.41.14 (22-Dec-2010)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
out/target/product/sugar-cubieboard2/obj/PACKAGING/systemimage_intermediates/unsparse_system.img: 1488/32768 files (0.0% non-contiguous), 99678/131072 blocks
Install system fs image: out/target/product/sugar-cubieboard2/system.img
out/target/product/sugar-cubieboard2/system.img+out/target/product/sugar-cubieboard2/obj/PACKAGING/recovery_patch_intermediates/recovery_from_boot.p maxsize=548110464 blocksize=4224 total=402252512 reserve=5537664
le@le: /work/le/a20-android/android$
```

(4) 打包，生成镜像

A. 在 android 目录下运行命令：

pack

B. 在 lichee/tools/pack 目录下出现.img 镜像则代表编译成功

2. 每次修改源代码后编译

在 android 目录下依次执行命令：

source build/envsetup.sh

lunch 16

make -j8

pack

3. 局部编译

在 android 目录下依次执行命令：

source build/envsetup.sh

lunch 16

mmm 文件路径

注：文件路径下应存在 makefile 文件，例如 Android.mk

4. 编译 Android sdk

在 android 目录下依次执行命令：

```
source build/envsetup.sh
make update-api（当 framework 层的接口未改变则无需执行该条命令）
lunch sdk-eng
make sdk -j8
```

11 蓝牙 APP 前端

11.1 界面设计原则

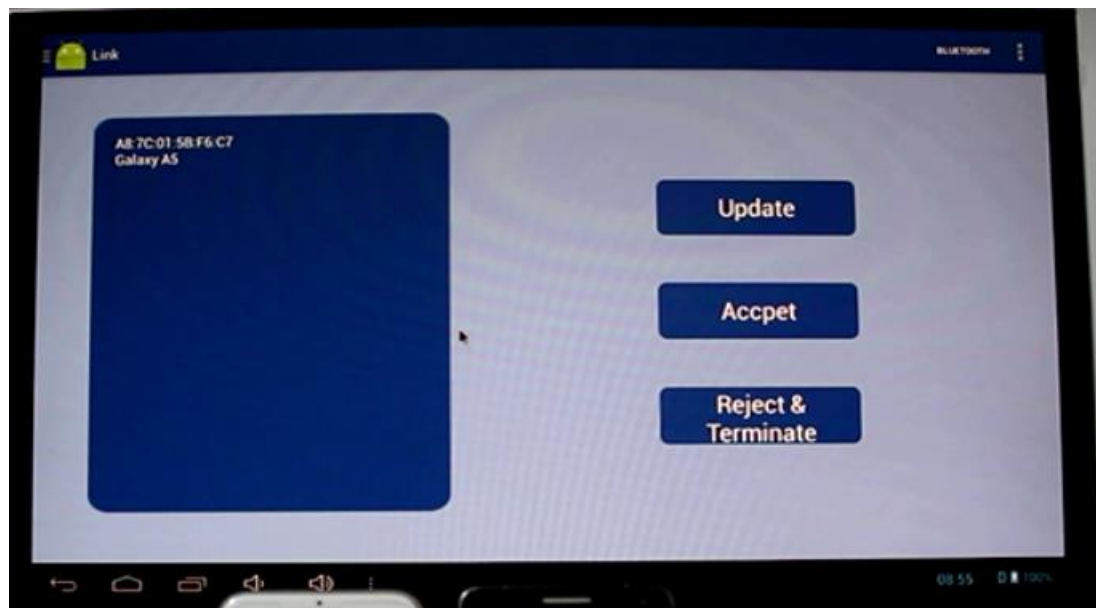
（1）易用性：考虑到用户群体的广泛性，界面应对用户友好，交互方式便捷易用，学习成本不宜高，故使用大色块+文字提示的扁平化设计，使得用户无需花费多余时间即可上手使用；

（2）简洁性：由于该项目为车载蓝牙系统，基于用户使用场景考虑，若堆砌元素过于繁杂容易分散用户注意力，色彩过于明艳或灰暗可能会影响用户使用心情，不宜正在开车的用户使用；故而界面应尽量简洁明了，色彩采用蓝白搭配，温和清新，有利于帮助用户保持平和良好的心态；

（3）逻辑清晰：由于有不同页面间的切换，而过多的页面“转场”则会使用户感到繁琐焦虑，故仔细考虑了不同页面间的逻辑关系，并增加了不同页面间的直接跳转，使得用户使用过程中更加方便快捷。

11.2 各界面设计详述

（由于蓝牙 APP 需要连接展示，故只能通过视频截屏方式取得界面图片，清晰度和美观度较低，敬请谅解）



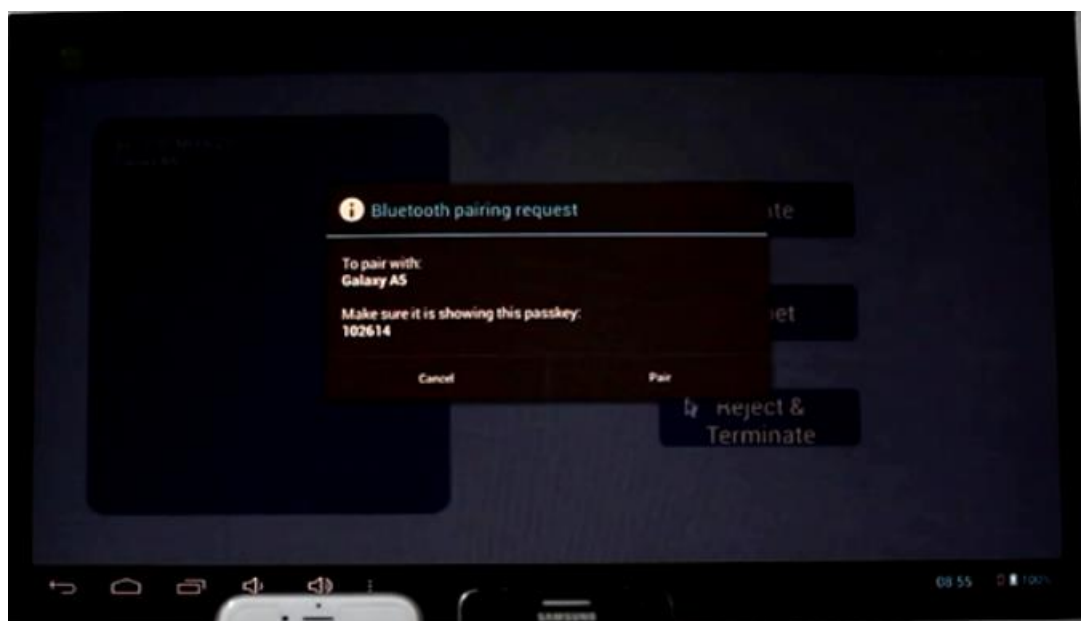
主界面左侧：连接列表展示；

右侧 Update：如连接过程中手机通讯录有变动，则可点击更新；

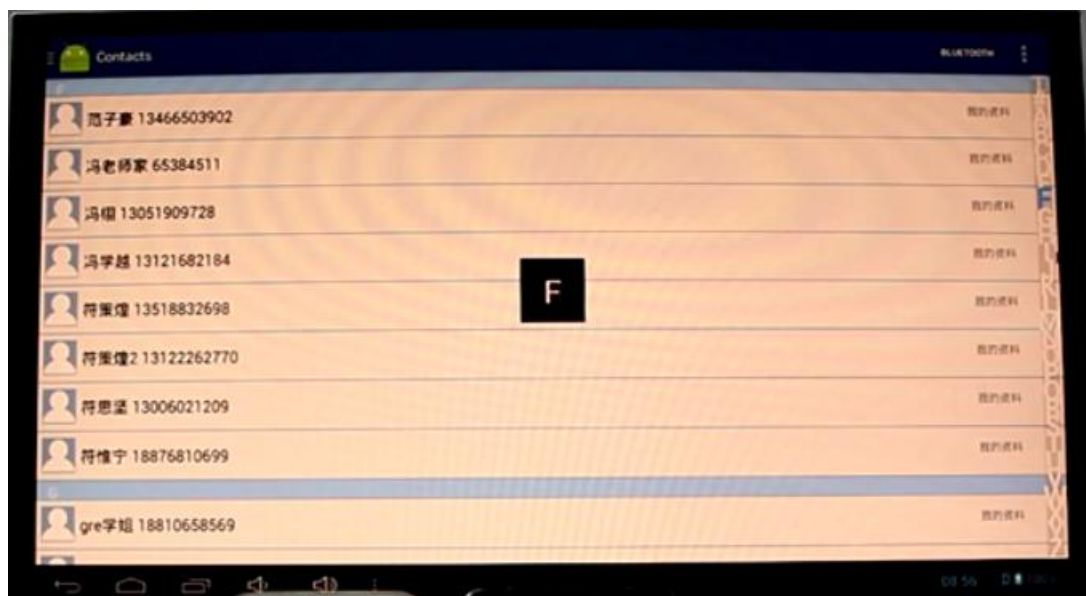
右侧 Accept：接听按钮，手机有电话进来时可点击接听；

右侧 Reject&Terminate：挂断和中断按钮，可挂断未接听和正在接听的电话；

右上角为蓝牙连接按钮：点击则会弹出对话框提示蓝牙配对连接；



配对成功后则会自动同步通讯录，并跳转；



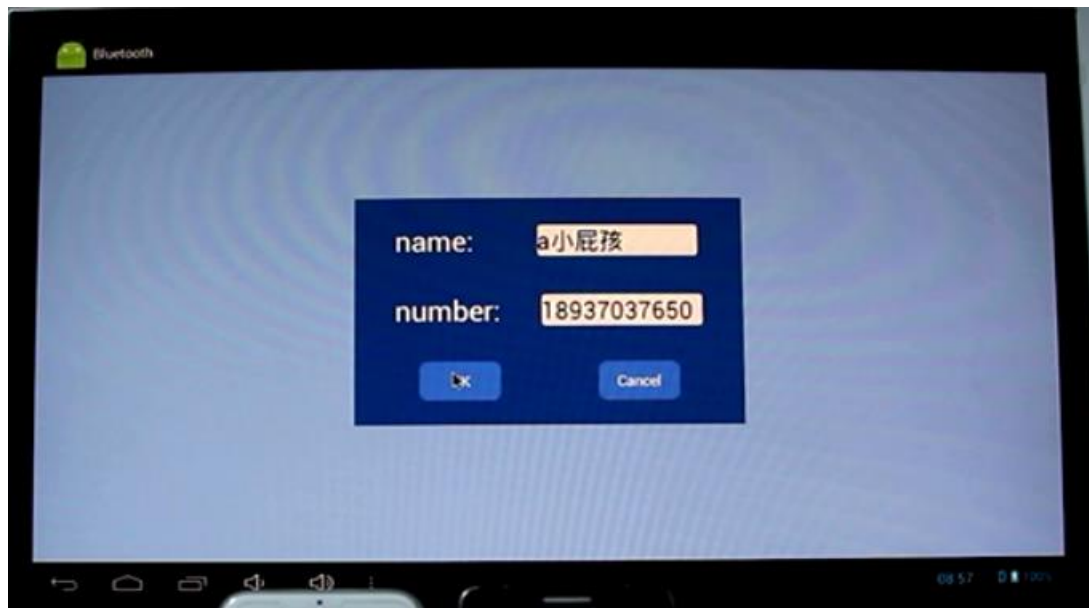
通讯录界面采用模仿微信界面设计，可通过右侧字母栏快速查找联系人，并跳转；



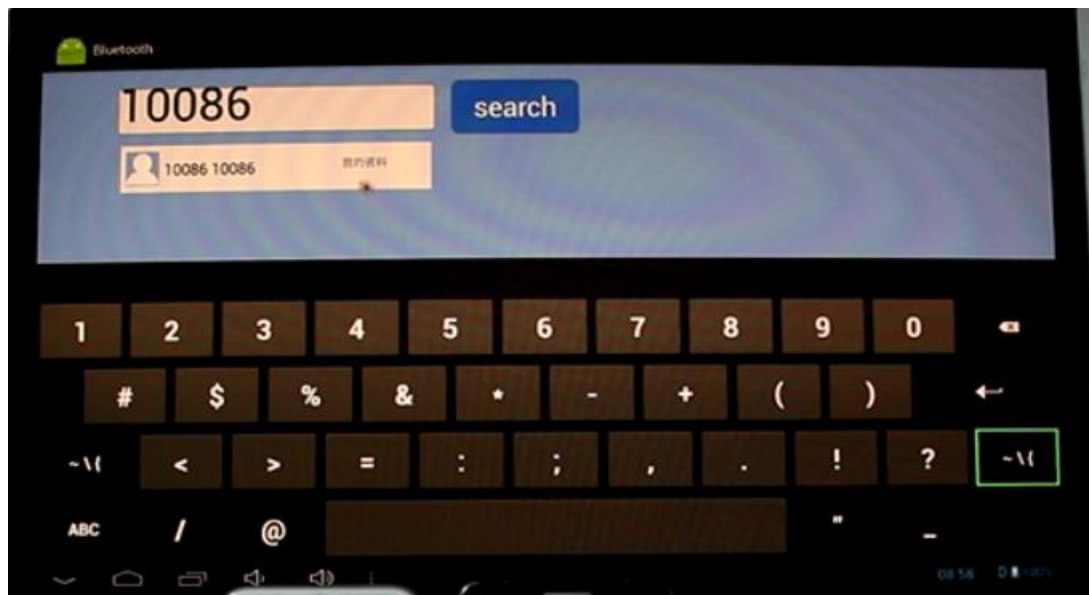
单击通讯人项目即可进行一系列操作，包括拨号，编辑，查找，新建等功能，并直接进行相应界面的跳转：

点击拨号：直接通过手机拨出；

点击编辑：跳转至编辑界面并可编辑通讯录保存，如下图：

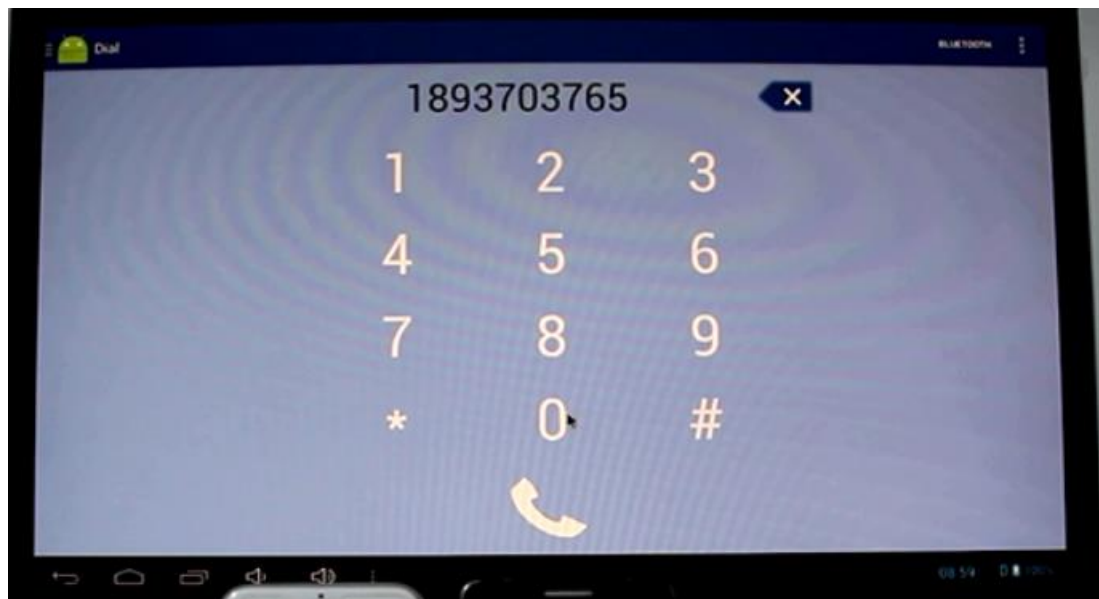


点击查找：跳转至查找界面，如下图：

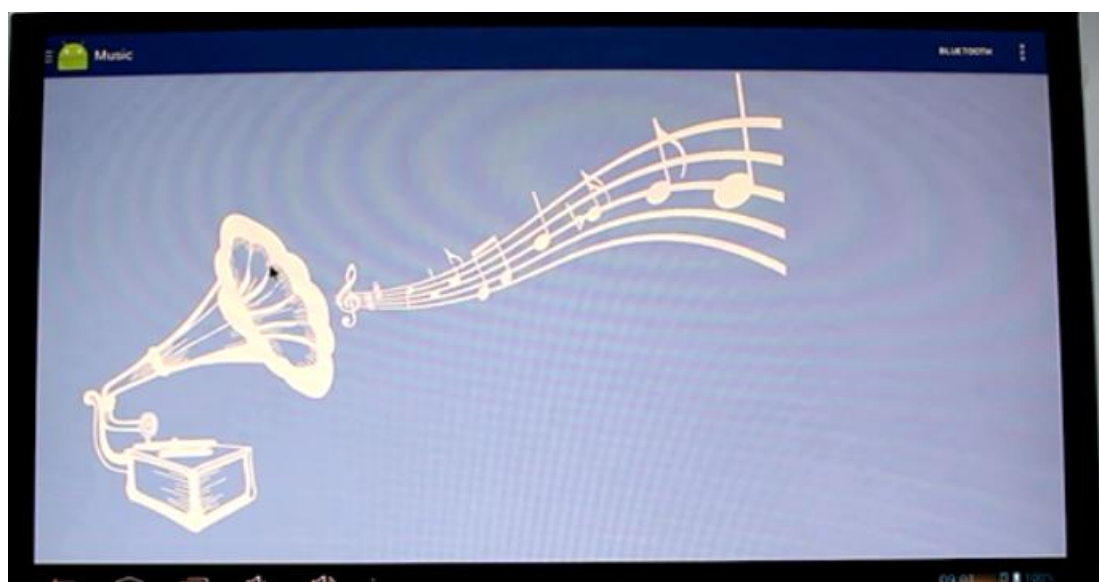


若在查找时直接点击查找结果则可直接跳转至拨号界面；

也可通过侧边抽屉栏跳转至拨号界面并进行拨号；



手动拨号界面，完整利用车载界面较大的空间，为用户提供更好的点击与观赏体验



音乐传输界面，美丽简洁的动态图案既不会吸引过多的注意力又有利于用户在开车时期放松心情。