**Dialer and calllog**

1. **Contacts数据库介绍**

在dailer和calllog的中，会经常用到联系人数据库的一些表。熟悉这些表及操作这些表有利于模块的学习。

1. **联系人的数据库文件的位置**

/data/data/com.android.providers.contacts/databases.contacts2.db

1. **数据库中重要的几张表**
2. contacts表

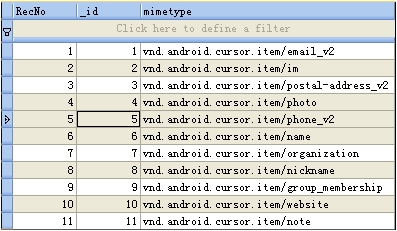
该表保存了所有的手机测联系人，每个联系人占一行，该表保存了联系人的ContactID、联系次数、最后一次联系的时间、是否含有号码、是否被添加到收藏夹等信息。

1. raw\_contacts表

该表保存了所有创建过的手机测联系人，每个联系人占一行，表里有一列标识该联系人是否被删除，该表保存了两个ID：RawContactID和ContactID,从而将contacts表和raw\_contacts表联系起来。该表保存了联系人的RawContactID、ContactID、联系次数、最后一次联系的时间、是否被添加到收藏夹、显示的名字、用于排序的汉语拼音等信息。

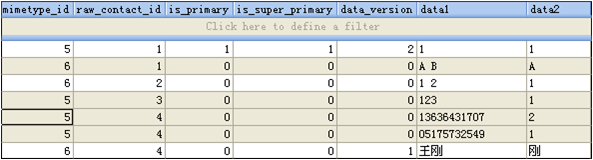
1. mimetypes 表

该表定义了所有的MimeTypeID，即联系人的各个字段的唯一标志。



1. data表

该表保存了所有创建过的手机测联系人的所有信息，每个字段占一行 ，该表保存了两个ID：MimeTypeID和RawContactID,从而将data表和  
raw\_contacts表联系起来。联系人的所有信息保存在列data1至data15中，各列中保存的内容根据MimeTypeID的不同而不同。如保存号码(MimeTypeID=5)的那行数据中，data1列保存号码，data2列保存号码类型(手机号码/家庭号码/工作号码等)。



1. calls 表

该表保存着通话记录的信息。表的主要字段说明：

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段类型 | 备注 |
| \_id | INTEGER | 记录标识 |
| number | TEXT | 电话号码 |
| date | INTEGER | 通话时间( 注意是一个整形 ) |
| duration | INTEGER | 电话时长 |
| new | INTEGER |  |
| name | TEXT | 联系人 |
| type | INTEGER | 电话状态(1 ：打进来的电话，2 ：打出的电话， 3 ：未接电话 ) |

1. **Dialer**
2. **主要java文件：**

Dialer即拨号盘,其主要的文件如下：

1. DialtactsActivity.java：这个文件是一个TabActivity，顾名思义就是用来控制点开idle界面的拨号按钮后所显示出来的那个4个tab(拨号，通话记录，通讯录，收藏)，这个文件与拨号和call log模块的维护虽然关系不大，但是看这个文件可以知道tab的默认设置和切换是怎么操作的，而且这个文件也是拨号和通话记录的入口。像setupContactsGroupTab和setupCallLogTab等函数就是用来初始化各个tab，setCurrentTab这个函数用来初始化第一次开机时候的tab和其他情况进入的时候需要显示的tab，比如挂断电话后进入的时候需要显示call log的tab等也是在这里设置的.
2. TwelveKeyDialer.java：这个是拨号盘使用的文件，所有拨号盘的东西全在这里，一个4500多行的文件，包含了拨号界面的构造，menu的构造，长短按的事件，IP呼叫，语音呼叫，特殊字符的功能，输入匹配等，拨号盘是否显示或者隐藏，判断是否两个SIM卡并显示不同的布局，主要作用还是打包数据发起一个call。
3. ContactsApplication.java ：这里有一个内部类和几个函数，在查询联系人信息的时候会被调用。如下：
4. DialpadContactInfo

此类用于存储联系人的信息，包括联系人名字首字母对应的数字的字符串以及一些其它用于后面进行联系人匹配的字符串，主要是判断输入匹配的。

1. setNameInfo(String sortkey)

将srortkey的拼音字母转换成对应的数字，然后存入对应的字符串数组

1. commpare(String input)

将input与DialpadContactInfo里的数字字符串进行匹配。从而判断input是否与联系人信息匹配，属于哪种匹配方法,具体匹配方法如下：

NUM\_MODE（号码匹配）；

FIRST\_PINYIN\_MODE（联系人名字拼音首字母匹配）；

ALL\_NAME\_MODE（名字拼音全名匹配）；

1. **布局**

布局文件是使用xml文件写死的，根据卡的状态来进行加载，在oncreate函数中，会根据卡1和卡2的有效情况进行布局，不同点是拨号的最下面怎么显示那个拨号按钮：这里会根据有卡，无卡，单双卡，卡1或者卡2无效进行不同的加载。所以布局文件只需要顺了xml一级一级的往下找，就能很清楚的知道怎么布局。如twelve\_key\_dialer.xml，这个是2个卡都有效的时候用的文件，根据它里面的LinearLayout和include的layout文件就能看到匹配框，输入框，12个数字按钮，拨号键的构造

1. **功能**

要看懂一个4500多行的代码，如果从头看到尾，一个一个分析，显然不容易理解，因此可以按照功能拆分成几个模块，先分析每个模块，然后再分析模块之间的联系,这也为以后增加或者删除某个功能提供了便利。

1. **class TwelveKeyDialer 所重载的函数**

该 activity 是拨号盘实现的主体框架，首先分析它重载的函数：

1. public void beforeTextChanged()；

该组件继承了TextWatcher，重载本函数，但是无内容

1. public void onTextChanged();

该组件继承了TextWatcher，重载本函数，但是无内容

1. public void afterTextChanged();

该组件继承了TextWatcher，重载本函数，后面新建一个textwatcher 实例与拨号盘输入框绑定，用于监听输入内容。本函数根据输入内容的不同进行相应的判断与处理

1. onCreate();
2. onCreateContextMenu()；

判断长按的view是联系人信息还是通话记录

1. onRestoreInstanceState()；

空函数

1. void onNewIntent()；

该函数只有launchMode为singleTop，当执行 [startActivity(Intent)](file:///F:\android\android-sdk-windows\docs\reference\android\app\Activity.html#startActivity(android.content.Intent))时，才会被调用。

本函数中，主要调用resolveIntent();

1. onPostCreate()；

该函数在调用[onStart()](file:///F:\android\android-sdk-windows\docs\reference\android\app\Activity.html#onStart()) 和 [onRestoreInstanceState(Bundle)](file:///F:\android\android-sdk-windows\docs\reference\android\app\Activity.html#onRestoreInstanceState(android.os.Bundle))后被调用

本函数中，主要实现TextChangedListener与mDigits的绑定

1. onResume()；
2. onWindowFocusChanged()；

函数内实现mDigits输入隐藏软键盘的作用

1. onPause()；
2. onDestroy()；
3. onCreateOptionsMenu()；

根据卡1和卡2的状态，选择增加子菜单

1. onPrepareOptionsMenu()；

根据相关条件，决定哪些子菜单应该显示

1. onClick()；

实现当绑定的view被点击后的操作

1. onKeyDown()；

当call实体键被按下后，起呼语音拨号

1. onKeyUp()；

当call实体键按下松开后，呼叫对应的号码

1. onLongClick()；

实现当绑定的view被长击后的操作

1. **导入通话记录**

在输入框无输入时，导入并显示通话记录，主要实现函数如下：

1. getCallLogCursor()

将导入的通话记录无重复的显示出来

1. getCallsInfo()

从数据库中导出通话记录，并保存。

1. getcalllognoreapeat()

去掉号码重复的通话记录

1. CallLogAdapter()

内容适配器，界面的布局与数据的绑定都在该类的getView()中实现。

1. **导入匹配的联系人(在w621代码中，并未使用此功能)**

根据输入框的输入内容，进行联系人的匹配并显示，主要实现函数与类如下：

1. setQueryContacts(String input);

根据输入的内容进行匹配，然后显示匹配到得联系人信息。

1. LoadQueryContactsAdapter

内容适配器，界面的布局与数据的绑定都在该类的getView()中实现。

1. GetAllContactsThread

一个线程类，该线程用于查询所有联系人信息，并进行相关处理，存入

ContactsApplication.allContacts中。

1. gatherContacts()

新建并启动一个GetAllContactsThread 线程

1. 备注：

这是一个相对独立的小系统，为外部提供的接口为（1）与（4）。在这个小系统里面有2个数据比较重要

1. ContactsApplication.allContacts。这个是定义在ContactsApplication的一个链表。该链表用于存储查询到得所有联系人信息。
2. queryContacts：这个链表是用于存储匹配到得联系人信息（在（1）内获取），然后与内容适配器进行绑定
3. **导入匹配的通话记录与联系人**

当输入框有输入时，进行通话记录与联系人匹配，并显示出来，主要实现函数与类如下：

1. getCallLogAndQueryContacts()

将输入值赋给minput，然后调用startQuery();

1. startQuery(String filter)

利用mQueryHandler.startQuery()查询联系人信息

1. QueryHandler. onQueryComplete()

当执行完联系人查询的任务后，回调该函数。函数将查询到得信息存入

mContactsCursor，并调用showMatchedContactsAndCalllogList();

1. showMatchedContactsAndCalllogList()

显示匹配好的通话记录与联系人信息

1. LoadCallLogAndQueryContactsAdapter

内容适配器，在getView()函数中，根据相关匹配原则刷选合适的通话记录与联系人信息，同时根据相关UI显示原则进行显示

1. **起呼一个call**

不管是输入号码或者是长按短按等操作，最后都是呼出call。呼出call的流程都是一样的，就是发出一个intent，代码为：

String num = getNum(number);

Intent intent = new Intent(Intent.ACTION\_CALL\_PRIVILEGED);

intent.setData(Uri.fromParts("tel", num, null));

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

startActivity(intent);

这是发起一个call所必须的，如果是单卡的话，就没有什么

问题了，从布局到数据导入再到字符输入和匹配再到发

Call 的intent ，基本上拨号盘就是做这些事。

1. **快速拨号**

长按数字的话，长按1是呼叫语音信箱，长按2到9是快速拨号，这些都是在设置里面设置的，并保存在不同的数据库中。主要函数如下：

1. callVoicemail();

拨打语音信箱号码

1. Speedcall(int number);

实现快速拨号

1. getSpeedNum(int padNum);

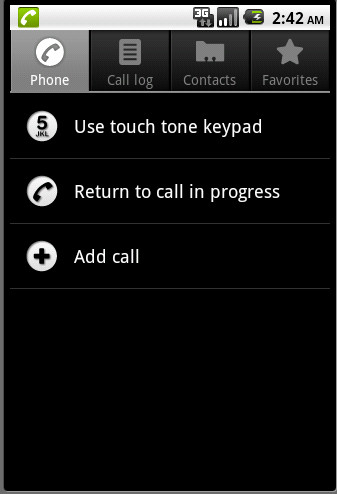
获取数字键对应的电话号码

1. **showDialpad**

当用户正在进行通话的时候，如果用户再次执行拨号界面，系统会弹出一个新的ui，即dialpad.让用户进行如下选择:

1. 返回通话中；
2. 新建一个通话
3. 使用触摸键盘

具体界面如下：



主要实现函数如下：

1. showDialpadChooser（boolean enabled）

根据传入值进行界面的显示或隐藏

1. boolean dialpadChooserVisible()

判断dialpad的显示状态

1. class DialpadChooserAdapter

内容适配器。界面的布局都在此类中完成

1. returnToInCallScreen(boolean showDialpad)；

返回通话界面

1. **输入字符的处理**

拨号盘除了拨号功能外，还有一些特殊字符特殊操作的处理，比如常见的输入\*#06#查询MEID号，不常见的如\*#351#查询呼叫限制状态等都是类似\*#数字#或者是\*#\*#数字#\*#\*等特殊字符串。这类功能的实现是：每次输入字符，都会调用到函数afterTextChanged，在这个函数里面会调用下SpecialCharSequenceMgr.lava里面的handleChars（）来判断，如果是特殊字符会截取中间的数字发intent出去实现相应的功能，同时clean掉输入框。不是的话则会用该字符去匹配记录以便于用户操作。对于匹配，如果匹配的数据有问题，那就去查看这个文件中的startQuery和QueryHandler函数，看是不是查询的条件和路径出错了，没错就看查询完之后的操作如setQueryContacts或者是onQueryComplete等函数，看查询后的又是对数据怎么操作的。这里提到的函数能处理的是特殊字符的操作和数据查询问题.

1. **对于双卡的处理**

这里写的双卡的东西在拨号盘和call log里面都是一样的。不管是拨号盘和call log，在操作的时候都是与通话设置和call相关联的。既然分了卡1和卡2，那比如语音信箱的设置就有卡1的和卡2的，快速拨号也有卡1设置的和卡2设置的，拨号的时候也需要选择是卡1还是卡2.这里有2点需要注意：一是需要与通话设置那边沟通好，如卡1和卡2的一些设置和属性放在什么地方，怎么去调用等等；二是我们在双卡手机中传数据出去的时候要表明这个数据是卡1还是卡2的，这里也要做好，不然很容易混乱。这个标记起来很容易，就是在intent里面增加2个数据，就是：

intent.putExtra("Subscription", 0)；

intent.putExtra("subId", 0)；

这里的0表示卡1，如果是想用卡2的，就把这2个0换成2个1。

1. **按键音功能**：

private ToneGenerator mToneGenerator

定义一个声音产生器

private Object mToneGeneratorLock = new Object()

监视器对象锁

private static final int TONE\_LENGTH\_MS = 150

声音播放时间

private static final int TONE\_RELATIVE\_VOLUME = 80

相对音量

private boolean mDTMFToneEnabled;

（1）首先初始化声音产生器 mToneGenerator,代码如下，一般在onCreate()或者onResume()中执行此段代码

synchronized(mToneGeneratorLock)

{

if (mToneGenerator == null)

{

try {

mToneGenerator = new ToneGenerator(DIAL\_TONE\_STREAM\_TYPE,

TONE\_RELATIVE\_VOLUME);

setVolumeControlStream(DIAL\_TONE\_STREAM\_TYPE);

}

catch (RuntimeException e)

{

Log.w(TAG, "Exception caught while creating local tone generator: " + e);

mToneGenerator = null;

}

}

（2）定义playtone函数，实现声音，当某按键需要发声音时，调用此函数

void playTone(int tone)

{

if (!mDTMFToneEnabled)

{

return;

}

AudioManager audioManager = (AudioManager) getSystemService(Context.AUDIO\_SERVICE);

int ringerMode = audioManager.getRingerMode();

if ((ringerMode == AudioManager.RINGER\_MODE\_SILENT)

|| (ringerMode == AudioManager.RINGER\_MODE\_VIBRATE))

{

return;

}

synchronized(mToneGeneratorLock)

{

if (mToneGenerator == null)

{

Log.w(TAG, "playTone: mToneGenerator == null, tone: "+tone);

return;

}

mToneGenerator.startTone(tone, TONE\_LENGTH\_MS);

}

}

1. **匹配记录的上下文菜单功能**

当长按匹配的联系人信息或通话记录时，弹出上下文菜单

在oncreate()函数中执行“ls.setOnCreateContextMenuListener(this);”，然后再重载onCreateContextMenu()，onContextItemSelected()即可实现:

1. onCreateContextMenu()：

判断长按的view是联系人信息还是通话记录，然后选择调用(2)或(3)

1. addMenuForCalllog():

为通话记录的view添加上下文菜单

1. addMenuForContacts():

为联系人的view添加上下文菜单

1. onContextItemSelected()

当某一子菜单被选择的时候，根据相应的判断，进行相应的操作

1. **Menu菜单**
2. onCreateOptionsMenu(Menu menu)

根据卡1和卡2的状态，选择增加子菜单

1. onPrepareOptionsMenu()

根据相关条件，决定哪些子菜单应该显示

1. onOptionsItemSelected(MenuItem item)

当子菜单被选中后，进行相应的操作

1. **对话框**

在进行一些操作时，会弹出对话框。主要函数如下：

1. onCreateDialog(int id);

根据id判断应该弹出哪个Dialog;

1. onPrepareDialog(int id, Dialog dialog);
2. IPshowdialog(Context context);
3. IPGSMshowdialog(Context context);
4. IPWCDMAshowdialog(Context context)
5. SpeedcallUnenable(Context context);
6. NoneSpeedcallNum(Context context);
7. **保存正在编辑号码**

拨号界面在拨号的EditText中显示该号码并编辑！此时编辑过程中出现锁屏解锁是编辑的数据不保存！原因是我们获取Intent的数据时会在onResume中获取！  
锁频后解锁时发出的Intent和之前跳转过来的是同一个！所以编辑数据不会保留会去INTENT中的数据。解决此问题的主要函数为：

1. resolveIntent();

根据相应的判断，来获取之前的编辑内容。

被调用函数：onCreate();

onResume()；

1. **震动功能**
2. queryHapticFeedbackEffectsEnabled();
3. HapticFeedbackEffects();
4. **其它**
5. phoneIsInUse(int sub)

判断电话是否正在被使用

1. phoneIsCdma(int sub)

判断电话是否是CDMA制式

1. phoneIsOffhook(int sub)

判断电话是否正在通话中

1. isDialerEnabled(int sub)

判断是否可以正常拨号

1. checkCardIfValid(int sub)

判断卡是否正常

1. onWindowFocusChanged(boolean hasFocus)

函数内实现mDigits输入隐藏软键盘的作用

1. setupKeypad()

为各个按钮绑定相对应的事件

1. setFormattedDigits()

将数据装换成标准号码格式，并在mDigits上显示出来

1. updateDialAndDeleteButtonEnabledState()

更新拨号按键与删除按键的状态

1. queryLastOutgoingCall()

查询上次拨出的电话号码并返回

1. **call log**
2. **主要java文件**

Call log的主要文件如下：

1. **RecentCallsListActivity.java:** 这个是call log的文件，包括call log界面的构造，menu的构造，数据的存储和查询，记录的填充显示，记录的长按和短按后的事件，以及发起一个call的准备等都在这个文件里面，call log大概做2件事：一是保存记录，同时需要把这些记录与联系人进行交互并且很好的显示出来。二是通过调用call和短信的接口，将记录里面的号码和姓名等参数传给call或者短信，就是提供一些便捷的操作让用户从call log中发起一个呼叫或者短信
2. **CallDetailActivity.java**: 通话记录的详细界面在这个类中实现
3. **GroupingListAdapter**: 一个继承了baseadapter的抽象类，实现了通话记录的合并分组，在 RecentCallsListActivity.java中有一个RecentCallsAdapter继承该类。其主要函数介绍如下：
4. changeCursor(Cursor cursor):

改变mcursor的值

1. getCursor()

返回mcursor

1. findGroups()

新建组，将调用addGroups(mCursor)；

1. addGroup()；

增加组

1. getCount()

获取显示的item的个数

1. obtainPositionMetadata(PositionMetadata metadata, int position);

判断位于position的item是那种类型，并将相关数据存入metadata中

1. toggleGroup(int position)

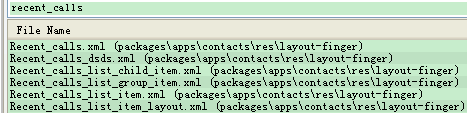
如果该位置的item是组头，或展开组，显示组的成员

1. getView(int position, View convertView, ViewGroup parent)

view构建函数，是整个adapter的核心

1. **ChooseCallLogToDeleteActivity**：选择删除通话记录的详细界面在这个类中实现
2. **布局**

通话记录模块的xml文件不多，但是都是嵌套在一个地方使用，所以跟着xml把顺序理好就很清楚其布局结构了：



注：1. recent\_calls.xml是单卡的布局

2. recent\_calls\_dsds.xml是双卡文件的布局。

3. list \_group是合并记录后的布局

4. List\_child是点开合并的记录后它里面的每一条的布局

5. List\_ltem是单个记录的布局

6. call\_detail.xml是点开单个记录，查看详细情况的时候，

使用布局文件。

1. **功能**
2. **通话记录的查询**

每当需要重新查询通话记录的时候，直接调用startQuery()即可：

1. startQuery()：根据cardSpinner（全部/卡1/卡2）的选择与typeSpinner（全部/已拨/已接/未接）的选择查询相对应的通话记录；
2. CatchingWorkerHandler. onQueryComplete (int token, Object cookie, Cursor cursor);

当查询完通话记录后，回调本函数。函数内调用callsAdapter.changeCursor(cursor)，将最新查询的记录cursor赋给mdapter;

1. **通话记录的显示与保存**

这主要是由RecentCallsAdapter这个adapter类实现，该类继承了GroupingListAdapter，该类的简要解释如下：

1. onClick(View view)

继承View.OnClickListener，然后重载的函数，后面将每个通话记录的callView与之绑定，实现每当点击callview图标时，根据相应判断条件，实现对该记录号码的呼叫。

1. onPreDraw()

继承ViewTreeObserver.OnPreDrawListener，然后重载该函数。并在bindView()中将实例mPreDrawListener与view绑定，鉴定第一次绘制UI，然后触发本函数。在本函数内，新建一个线程用通话记录里的号码去联系人的数据库里进行匹配

1. RecentCallsAdapter()

构造函数，用于初始化和载入相关资源

1. onContentChanged()`
2. setLoading(boolean loading)

设置mloading的值，mloading的值用于判断是否为空记录

1. isEmpty()

根据mloading的值，判断通话记录是否为空

1. getContactInfo(String number)

通过传入的号码 number，在mContactInfo查询对应号码的联系人信息并返回。

1. startRequestProcessing()

开始匹配查询线程

1. stopRequestProcessing()

终止匹配查询线程

1. clearCache()

清除mContactInfo中的所有联系人信息

1. updateCallLog(CallerInfoQuery ciq, ContactInfo ci)

将联系人的信息ci更新到ciq号码的通话记录里，实现通话记录的更新

1. enqueueRequest(String number, int position,String name, int numberType, String numberLabel)

将传入的通话记录信息存入到匹配链表mRequest中，等待匹配

1. queryContactInfo(CallerInfoQuery ciq)

查询ciq号码对应的联系人信息，并根据对应的判断条件，存入mContactInfo

1. run()

继承Runnable接口，重载此函数。本函数用于循环处理mqueset,调用queryContactInfo进行联系人匹配

1. addGroups(Cursor cursor)

根据相应的判断条件，决定是否新建组

1. equalPhoneNumbers(CharArrayBuffer buffer1, CharArrayBuffer buffer2)

判断2个号码是否相同

1. newStandAloneView(Context context, ViewGroup parent)

新建一个单独view

1. bindStandAloneView(View view, Context context, Cursor cursor)

绑定view

1. newChildView(Context context, ViewGroup parent)

新建一个子view

1. bindChildView(View view, Context context, Cursor cursor)

绑定子view

1. newGroupView(Context context, ViewGroup parent)

新建一个组view

1. bindGroupView(View view, Context context, Cursor cursor, int groupSize,boolean expanded)

绑定组view

1. findAndCacheViews(View view)

为view载入布局文件

1. bindView(Context context, View view, Cursor c)

给view各个子部件绑定数据，并根据相应的判断条件是否可见

1. **通话记录的上下文菜单功能：**

在oncreate()函数中执行“getListView().setOnCreateContextMenuListener(this);”，然后再重载onCreateContextMenu()，onContextItemSelected()即可实现:

(ContextMenu menu, View view, ContextMenuInfo menuInfoIn)：

根据条件判断，选择性创建如下上下文菜单：

语音呼救号码；

1. 卡1 IP呼叫号码；
2. 卡2 IP呼叫号码；
3. IP呼叫号码；
4. 查看联系人
5. 发送名片
6. 呼叫之前编辑号码
7. 发送短信
8. 发送号码
9. 添加至联系人
10. 从通话记录中删除
11. 复制号码

onContextItemSelected(MenuItem item)：  
当某一子菜单被选择的时候，根据相应的判断，进行相应的操作

1. **Menu菜单**

onCreateOptionsMenu(Menu menu)；

如果记录不为空，则创建一个标题为“删除通话记录”的menu;

onOptionsItemSelected(MenuItem item)：

当点击了删除记录menu后，进入ChooseCallLogToDeleteActivity，选择删除哪些记录。

1. **实体键Call功能**
2. onKeyDown(int keyCode, KeyEvent event);

起呼一个语音拨号；

1. onKeyUp(int keyCode, KeyEvent event); callEntry(int position);

起呼焦点所在通话记录的号码

隐藏和显示拨号盘

void hideShowDialpad(){

if (mDialpad != null){

if(isDialpadShow){

isDialpadShow=false;

}else{

isDialpadShow=true;

}

updateShowDialpad(isDialpadShow);

}

}

void updateShowDialpad(boolean isShow){

Log.i(TAG,"updateShowDialpad isShow="+isShow);

if (mDialpad != null){

if(isShow){

((ImageView)mDialpadButton).setImageResource(R.drawable.ic\_in\_call\_touch\_dialpad\_close);

mdigit\_deleteButton.setVisibility(View.VISIBLE);

mDigits.setVisibility(View.VISIBLE);

mDialpad.setVisibility(View.VISIBLE);

isDialpadVisible=true;

if(!mLocalSpace.getText().toString().equals(""))

if(mLocalSpaceState==View.VISIBLE)

{

mLocalSpace.setVisibility(View.VISIBLE);

}

mDigits.requestFocus();

}else{

((ImageView)mDialpadButton).setImageResource(R.drawable.ic\_in\_call\_touch\_dialpad);

mDialpad.setVisibility(View.GONE);

mdigit\_deleteButton.setVisibility(View.GONE);

mDigits.setVisibility(View.GONE);

isDialpadVisible=false;

mLocalSpaceState=mLocalSpace.getVisibility();

mLocalSpace.setVisibility(View.INVISIBLE);

}

}

}

SIM卡的选择及配置

private Dialog IPGSMshowdialog(Context context)

{

String strNewIpNum = null;

final Context con = context;

strNewIpNum = Settings.System.getString(context.getContentResolver(),

"ip\_prefix\_num\_2");//Settings.System.IP\_CALL\_NUM\_GSM

Log.v(TAG,"wyh IPGSMshowdialog strNewIpNum ="+strNewIpNum+"mDigits.getText().toString()="+mDigits.getText().toString());

if(null != strNewIpNum && strNewIpNum.length()>0 ) {

ListView lv=new ListView(this);

ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,

//20110712 nhl change the layout add single line display

//android.R.layout.simple\_list\_item\_single\_choice, new IPNumber().getIPnumber(this, mDigits.getText().toString()));

R.layout.simple\_list\_item\_single\_choice\_ip, new IPNumber().getGSMIPnumber(this, mDigits.getText().toString()));

lv.setAdapter(adapter);

//list item can get the focus

lv.setItemsCanFocus(true);

//the radio button just choice single

lv.setChoiceMode(ListView.CHOICE\_MODE\_SINGLE);

//the first radio button is checked in the default

lv.setItemChecked(0, true);

IPString=adapter.getItem(0);

OnItemClickListener itemClick=new OnItemClickListener() {

public void onItemClick(AdapterView<?> adapterView, View view, int id,

long positon) {

//Track the number that radio button is checked

IPString=((TextView)view.findViewById(android.R.id.text1)).getText().toString();

}

};

lv.setOnItemClickListener(itemClick);

AlertDialog.Builder builder = new AlertDialog.Builder(context);

builder.setPositiveButton(R.string.StartIPCall,

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

//Set the EditText and start call

mDigits.setText(IPString);

Log.v(TAG,"wyh IPshowdialog dialButtonPressed ");

dialButtonPressed(2);//wyh

}

});

builder.setNegativeButton(R.string.CancelIPCall,null);

AlertDialog alert=builder.create();

alert.setTitle(R.string.ip);

alert.setView(lv);

return alert;

} else {

AlertDialog.Builder builder = new AlertDialog.Builder(con);

builder.setTitle(R.string.no\_ip\_prefix)

.setIcon(android.R.drawable.ic\_dialog\_alert)

.setMessage(R.string.alert\_ip\_prefix\_message)

.setNegativeButton(android.R.string.cancel, null)

.setPositiveButton(R.string.ip\_configure, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int which) {

Intent i = new Intent("android.intent.action.MAIN" );

// i.setClassName("com.android.phone", "com.android.phone.CdmaCallOptions");//GsmUmtsCallOptions

i.putExtra("SUBSCRIPTION\_ID",1);

i.putExtra("subId", 1);

i.setClassName("com.android.phone", "com.android.phone.GsmUmtsCallOptions");

try {

Log.v(TAG,"wyh IPshowdialog startactivity ");

con.startActivity(i);

} catch(ActivityNotFoundException e) {

e.printStackTrace();

}

}

});

AlertDialog alert=builder.create();

return alert;

}

}

//Begin wanghaishan 20121215 add for wcdma ip call

*private* Dialog IPWCDMAshowdialog(Context context)

{

String strNewIpNum = null;

final Context con = context;

strNewIpNum = Settings.System.getString(context.getContentResolver(),

"ip\_prefix\_num\_1");//Settings.System.IP\_CALL\_NUM\_GSM

Log.v(TAG,"wyh IPWCDMAshowdialog strNewIpNum ="+strNewIpNum+"mDigits.getText().toString()="+mDigits.getText().toString());

if(null != strNewIpNum && strNewIpNum.length()>0 ) {

ListView lv=new ListView(this);

ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,

//20110712 nhl change the layout add single line display

//android.R.layout.simple\_list\_item\_single\_choice, new IPNumber().getIPnumber(this, mDigits.getText().toString()));

R.layout.simple\_list\_item\_single\_choice\_ip, new IPNumber().getIPnumber(this, mDigits.getText().toString()));

lv.setAdapter(adapter);

//list item can get the focus

lv.setItemsCanFocus(true);

//the radio button just choice single

lv.setChoiceMode(ListView.CHOICE\_MODE\_SINGLE);

//the first radio button is checked in the default

lv.setItemChecked(0, true);

IPString=adapter.getItem(0);

OnItemClickListener itemClick=new OnItemClickListener() {

public void onItemClick(AdapterView<?> adapterView, View view, int id,

long positon) {

//Track the number that radio button is checked

IPString=((TextView)view.findViewById(android.R.id.text1)).getText().toString();

}

};

lv.setOnItemClickListener(itemClick);

AlertDialog.Builder builder = new AlertDialog.Builder(context);

builder.setPositiveButton(R.string.StartIPCall,

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

//Set the EditText and start call

mDigits.setText(IPString);

Log.v(TAG,"wyh IPWCDMAshowdialog dialButtonPressed ");

dialButtonPressed(1);//wyh

}

});

builder.setNegativeButton(R.string.CancelIPCall,null);

AlertDialog alert=builder.create();

alert.setTitle(R.string.ip);

alert.setView(lv);

return alert;

} else {

AlertDialog.Builder builder = new AlertDialog.Builder(con);

builder.setTitle(R.string.no\_ip\_prefix)

.setIcon(android.R.drawable.ic\_dialog\_alert)

.setMessage(R.string.alert\_ip\_prefix\_message)

.setNegativeButton(android.R.string.cancel, null)

.setPositiveButton(R.string.ip\_configure, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int which) {

Intent i = new Intent("android.intent.action.MAIN" );

// i.setClassName("com.android.phone", "com.android.phone.CdmaCallOptions");//GsmUmtsCallOptions

i.putExtra("SUBSCRIPTION\_ID",0);

i.putExtra("subId", 0);

i.setClassName("com.android.phone", "com.android.phone.GsmUmtsCallOptions");

try {

Log.v(TAG,"wyh IPshowdialog startactivity ");

con.startActivity(i);

} catch(ActivityNotFoundException e) {

e.printStackTrace();

}

}

});

AlertDialog alert=builder.create();

return alert;

}

}

//End wanghaishan