# Technical document

The Name： ZTE WirelessModem UserGuide of Android for RIL

The Number：

The Version： V0.7

Total pages: 99

(Including the cover)

| | |
|---|---|
| Prepared by | Jinhui Jiang |
| | Pingbo Luo |
| Audit by | |
| Countersign | |
| | |
| | |
| Standard | |
| Approve by | |

ZTE Corporation

# Revision History

| Version | Modified date | Description of the problem | Prepared by | Modify the contents |
|---|---|---|---|---|
| 0.0 | 2011.10.17 | Create this document | Yile Zhang Jinhui Jiang | Create this document |
| 0.1 | 2011.11.22 | The fuction of Voice channel configuration | Yile Zhang | Add the contents of 3.1 |
| 0.2 | 2012.01.22 | Add a description for android2.3 | Jinhui Jiang | Modify7.2.3.3 |
| 0.3 | 2012.03.05 | Add section 8,9 to CDMA branch and Add section 4 to WCDMA branch | Jinhui Jiang Yile Zhang | Add section 8,9 to CDMA branch<br>Add section 4 to WCDMA branch |
| 0.4 | 2012.03.06 | the function of AT commands sending in the layer of framework | Yile Zhang | Add section 5to WCDMA branch and add the function of AT commands sending in the layer of framework |
| 0.5 | 2012.05.16 | Add the setting of APN in the branch of CDMA in Android4.0 to send MMS | Jinhui Jiang | add the setting of APN in Android4.0's CDMA branch,and add section 6.4 |
| 0.6 | 2012.05.24 | Organize the document | Yile Zhang | add Part1: RIL Driver integration<br>add the part of FAQ |
| 0.7 | 2012.07.17 | 1 The new added functions on the platfrorm of Samsung's smdkv210 and NVIDIA's cardhu14r7 in the branch of CDMA on the android4.0 OS<br>2 Modify and improve the document | Pingbo Luo | Part 2.4 Modify the sending of casecade English SMS in android4.0<br>Part 6.3.3 Modify and complete the setting of APN<br>Part 7.2 The function of call Forwarding and waiting in the branch of CDMA in Android4.0<br>Part 3.1 Modify the Saving of the Chinese contacts in the SIM card |

Note: In this file, the italics are the code, and the italics written in red indicate that the code should either be added or deleted. If there is a "+" before the code, it means that the code should be added. If there is a "-" before the code, it means that the code should be deleted.

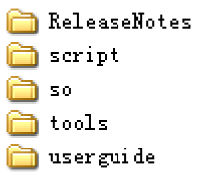
# Contents

# Part I: RIL Driver integration

## 1 The directory structure of ril driver package

### 1.1 Directory Structure

The directory structure for ril driver package of our corperation is as follows.



ReleaseNotes
script
so
tools
userguide

### 1.2 File Specification

The files description in driver package:

/ReleaseNotes Modify records and notifications

ReleaseNotes.txt

/script the script for data connection

init.gprs-pppd

ip-up-ppp0.c

ip-down-ppp0.c

/tools debugging tools

The tool to send AT commands

The tool to catch BP log

The tool to catch AP log

/userguide

Userguide of RIL adapter for Android System V0.x.pdf

ZTE WirelessModem UserGuide of Android for RIL V0.x.pdf

## 2 Linux kernel configuration

### 2.1 Add modem-driver

Adding modem-driver needs to configure the android kernel system as below:

```
cd kernel
make menuconfig
```

device drivers--->usb support--->usb serial converter support
Select the following components:
USB driver for GSM and CDMA modems

## 2.2 Add the VID and PID for Device

Find the kernel source file: option.c ( Usually，the path is: ..\kernel\drivers\usb\serial\option.c).
Find the following code which is blue in the source file, and then add the PID and VID which is written in red:

*static struct usb_device_id option_ids[]* = {
*{ USB_DEVICE(0x19d2, 0x1301) },*
0x19d2 is the ID of ZTE company
0x1301 is the ID of MF206 device
The ID of the above is depended on specific modem, and different modem has different ID.

## 2.3 Add ppp Components

The networking capability of RIL-driver uses ppp protocol to create data connection in underly, so we need to configure the kernel in order to surport ppp protocols. The configuration is shown as below:

cd kernel
make menuconfig
device drivers--->network device support--->ppp surport
Select the following components:
ppp filtering
ppp support for async serial ports
ppp support for sync tty ports
ppp deflate compression
ppp BSD-compress compression

# 3 Integrated driver files

## 3.1 Integrated dial-up script

**1 The files needed to copy**

Copy the files named ip-up-ppp0.c and ip-down-ppp0.c to ../external/ppp/android.
Create the directory of /third_part/zte under the directory of device.
Copy the file named init.gprs-pppd to the path of ../device/third_part/zte.
Copy the file named libreference-ril.so to the path of../device/third_part/zte.

**2 Modify the file: external/ppp/android/Android.mk**

```
diff --git a/android/Android.mk b/android/Android.mk
```

```
index 25c4c58..b4c48da 100644
--- a/android/Android.mk
+++ b/android/Android.mk

@@ -23,3 +23,21 @@
 LOCAL_MODULE := ip-up-vpn
  LOCAL_MODULE_PATH := $(TARGET_OUT_ETC)/ppp
  include $(BUILD_EXECUTABLE)
+
+include $(CLEAR_VARS)
+LOCAL_SRC_FILES := ip-up-ppp0.c
+LOCAL_SHARED_LIBRARIES := libcutils
+LOCAL_MODULE := ip-up-ppp0
+LOCAL_MODULE_PATH := $(TARGET_OUT_ETC)/ppp
+LOCAL_MODULE_TAGS := optional
+include $(BUILD_EXECUTABLE)
+
+include $(CLEAR_VARS)
+LOCAL_SRC_FILES := ip-down-ppp0.c
+LOCAL_SHARED_LIBRARIES := libcutils
+LOCAL_MODULE := ip-down-ppp0
+LOCAL_MODULE_PATH := $(TARGET_OUT_ETC)/ppp
+LOCAL_MODULE_TAGS := optional
+include $(BUILD_EXECUTABLE)
```

**3 Package the file into system**

Find the.mk file which is called at the time of system compiler, and usually this file located in the directory of /device/xxx/yyy. Here, xxx is indicates the manufacturer of platform provides, yyy represents a particular platform model of this manufacturer, for example, on the Freescale's imx51x platform, the file is located in directory device/fsl/imx5x. But sometimes this file is under the directory of vendor/xxx/yyy, for example, the directory is /vendor/nvidia/harmony on the nvida's harmony platform. Ordinarily, the naming scheme of this .mk file is whether "Platform name.mk" or "AndroidBoard.mk", for example, the file name is imx5x.mk for the Freescale's imx51x platform and AndroidBoard.mk for the nvida's harmony platform.

Add the following code in the file:

```
+ PRODUCT_PACKAGES += \
+       ip-up-ppp0 \
+       ip-down-ppp0
+ PRODUCT_COPY_FILES += \
+       device/third_part/zte/init.gprs-pppd:system/etc/init.gprs-pppd
+       device/third_part/libreference-ril.so:system/lib/librefernce-ril.so
```

Note：When the system has been compiled, you can find the file named init.gprs-pppd in the directory of system/etc/ which is under correspond platform in "out" directory. And you can find the files of ip-up-ppp0 and ip-down-ppp0 in the directory of system/etc/ppp which is also under correspond platform in “out” directory. The file libreference-ril.so can be found in the directory of system/lib/ which is under the out directory’s correspond platform too. If the files are not found, it indicates that the relevant files are not packaged into the System Image. Please check the configuration to make sure whether it is correct.

## 3.2 Modify the System Configuration

**1 Modify the Permissions of File: init.gprs-pppd**

Modify the code as follows:

Path: system/core/ include/private/android_filesystem_config.h

```
diff –git a/include/private/android_filesystem_config.h
b/include/private/android_filesystem_config.h

--- a/include/private/android_filesystem_config.h
+++ b/include/private/android_filesystem_config.h
@@ -158,6 +158,7 @@
static struct fs_path_config android_files[] = {
   { 00440, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/dbus.conf" },
   { 00440, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluez/main.conf" },
   { 00440, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluez/input.conf" },
+ { 00777, AID_ROOT, AID_SHELL, "system/etc/init.gprs-pppd" },
   { 00440, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluez/audio.conf" },
                          /* allow read-write to device-specific configuration files */
   { 00660, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluez/*" },
```

**2 Modify the Permissions of File: devices.c**

No matter the version is Android2.2 or earlier, modify the code as follows:

Path：system/core/init/devices.c b/init/devices.c

```
diff --git a/init/devices.c b/init/devices.c

index 0727974..94f8529 100644
--- a/init/devices.c
+++ b/init/devices.c
@@ -155,7 +155,11 @@
static struct perms_ devperms[] = {
      { "/dev/ts0710mux",    0640,   AID_RADIO,    AID_RADIO,    1 },
      { "/dev/ppp",          0660,   AID_RADIO,    AID_VPN,      0 },
      {"/dev/tun",           0640,   AID_VPN,      AID_VPN,      0 },
      {"/dev/video0",        0660,   AID_ROOT,     AID_CAMERA, 1 },
     +{"/dev/ttyUSB0",       0640,   AID_RADIO,    AID_RADIO,    0 },
     +{ "/dev/ttyUSB1",      0640,   AID_RADIO,    AID_RADIO,  0 },
     +{"/dev/ttyUSB2",       0640,   AID_RADIO,    AID_RADIO,    0 },
     +{ "/dev/ttyUSB3",      0640,  AID_RADIO,     AID_RADIO,  0 },
     +{ "/dev/ttyUSB4",      0640,  AID_RADIO,     AID_RADIO,  0 },
      { "/dev/snd/",          0664,  AID_SYSTEM,   AID_AUDIO,   1 },
    { NULL, 0, 0, 0, 0 },
};
```

No matter the version is Android2.3 or above, modify the code as follows:

Modify the file: system/core/rootdir/ueventd.rc, add the following to the file：

```
/dev/bus/usb/*          0660      root          usb
+/dev/ttyUSB0           0660      radio         radio
+/dev/ttyUSB1           0660      radio         radio
+/dev/ttyUSB2           0660      radio         radio
+/dev/ttyUSB3           0660      radio         radio
+/dev/ttyUSB4           0660      radio         radio
```

**3 Modify the File: system/core/init/property_service.c**

```
diff --git a/init/property_service.c b/init/property_service.c

index f3f044f..4718b00 100644
--- a/init/property_service.c
+++ b/init/property_service.c
@@ -59,6 +59,7 @@
 struct {
          { "net.usb1.",          AID_RADIO,      0 },
          { "net.rmnet0.",        AID_RADIO,      0 },
          { "net.gprs.",          AID_RADIO,      0 },
         +{ "net.ppp0.",          AID_RADIO,      0},
          { "net.ppp",            AID_RADIO,      0 },
          { "ril.",               AID_RADIO,      0 },
          { "gsm.",               AID_RADIO,      0 },
@@ -93,9 +94,8 @@
struct {
          unsigned int uid;
          unsigned int gid;
  } control_perms[] = {
-         { "dumpstate",AID_SHELL, AID_LOG },
-         { "pppd_gprs",AID_RADIO, AID_RADIO },
-         {NULL, 0, 0 }
+         {"pppd_gprs", AID_RADIO, AID_LOG },
+         {NULL, 0, 0 }
  };
  typedef struct {
```

**4 Modify the File: init.rc**

Path：(In general，the path is ../system/core/rootdir/init.rc. Refer to "Note" in page 11)

```
diff --git a/rootdir/init.rc b/rootdir/init.rc
@@ -265,7 +271,7 @@
service nexus /system/bin/nexus
service debuggerd /system/bin/debuggerd
-service ril-daemon /system/bin/rild
```

```
    ###add rild service,-d expressed the module receives the port parameters of AT services###
    ### -u indicate the Port of DATA###
    +service ril-daemon /system/bin/rild   -l   /system/lib/libreference-ril.so
        --   -d   /dev/ttyUSB1   -u /dev/ttyUSB3
        socket rild stream 660 root radio
        socket rild-debug stream 660 radio system
        user root

    ### Add mechanism of restarting the rild ###
    +on property:ril.reset.rild=1
    +stop ril-daemon
    +start ril-daemon
    +setprop ril.reset.rild      0

    ###Add mechanism of resetting the modem, ResetModem() is the function to realize the
resetting of modem by user themselves###
    +on property:ril.reset.modem=1
    +/system/bin/ResetModem:

@@ -329,6 +335,12 @@
service pbap /system/bin/sdptool add --channel=19 PBAP
          disabled
          oneshot
     ###Add pppd_gprs service###
     +service pppd_gprs /etc/init.gprs-pppd
     +user root
     +group radio cache inet misc
     +disabled
     +oneshot
     +
     service installd /system/bin/installd
     socket installd stream 600 system system
```

## 5 The Adapter of Tools for Upgrading and Log-catching

Modify the file of init.rc as follows:

```
setprop net.tcp.buffersize.umts      4094,87380,110208,4096,16384,110208
setprop net.tcp.buffersize.edge      4093,26280,35040,4096,16384,35040
setprop net.tcp.buffersize.gprs      4092,8760,11680,4096,8760,11680

+ setprop service.down.firmware    false
+ setprop service.ap.infoget    false
+ setprop service.bp.infoget    false

…………………
```

```
on property:persist.service.adb.enable=1
    start adbd

on property:persist.service.adb.enable=0
    stop adbd

+on property:service.down.firmware=true
+       stop ril-daemon
+       chmod 777 /dev/ttyUSB0
+       chmod 777 /dev/ttyUSB1
+       chmod 777 /dev/ttyUSB2
+       chmod 777 /dev/ttyUSB3
+       chmod 777 /dev/ttyUSB4
+on property:service.down.firmware=false
+       start ril-daemon

+on property:service.ap.infoget=true
+       chmod 777 /dev/log/radio
+       chmod 777 /dev/log/system
+       chmod 777 /dev/log/events
+       chmod 777 /dev/log/main

+on property:service.bp.infoget=true
+       chmod 777 /dev/ttyUSB0
+       chmod 777 /dev/ttyUSB1
+       chmod 777 /dev/ttyUSB2
+       chmod 777 /dev/ttyUSB3
+       chmod 777 /dev/ttyUSB4
```

NOTE: Some manufacturers maintain their own file of init.rc, so modify init.rc in this directory may not take effect, and usually this file is under the directory of /device/xxx/yyy or vendor/xxx/yyy. xxx indicates the manufacturer of platform provides, yyy represents a particular platform model of this manufacturer, for expample, on the Freescale's imx51x platform, the directory is located in directory device/fsl/imx5x.

# 4 The Configuration and Use of RIL

## 1 CDMA modem without SIM card, close RF

When the SIM card not inserted into the CDMA modem, it will search the network according to roaming list incessantly. In this case, the modem's power consumption will increase too. To solve this problem, we retained the software configuration items which will close RF at the state of non-card in the RIL. When the user set the value of property ril.sim.absent.do as close-antenna in the system, the RF will be closed if the SIM is found absence.

**2 Detect remaining number times of PUK and PIN**

Android system cann't get the initial value of the remaining number times of PUK and PIN by default. In other words, when the user input PIN, the remaining retry times of PUK and PIN is unable to get. At present, we will detect the initial value of PIN and PUK at the time of boot-up, and write the relevant information to the property value of *ril.sim.pin.retry.numbers* and *ril.sim.puk.retry.numbers*. The application in the upper layer can display the times in UI based on these two property values.

**3 The setting of serial prots and USB interface**

At present, the modem of our company supports the mode of USB access and serial ports access. In default, we use the USB access. If the users need to use the USB access + serial ports access, you need to set the the device path for serial ports, which is achieved by adding the property value of ril.serial.device.path as a particular serial ports at the time of system built.

**4 The function of CDMA modem for SIM card hot-unplug**

For the CDMA modem, when the SIM card is hot-unplug, the display of imformation about signal strength and network registration status are normal as before because the information of the SIM card has been retained to the modem. Some users consider that this is a BUG. If you want to avoid this, please reset the modem once detect the message about that the SIM card is pulled out.

**5 The function of writing the contacts into the SIM card on CDMA branch**

In the android native code, we need to set the storage ID when writing the contacts into the SIM card, and this ID incates that the contact information is written to a certain storage location on the SIM card. At present, the way to write is provided in the RIL. If the value of ID which is set by user is -1, the contact information is written into the SIM card's first empty location.

# 5 Debugging Method

You can type "adb shell" in the command window to enter ADB, and then you can input the following command to catch the LOG for analysis:

```
$ netcfg
$ logcat –b radio –v time
$ getprop
$ logcat –v time –s pppd
$ logcat –v time
$ dmesg
```

# Part II: Modify the android system for WCDMA modem's adaption

## Special Explanation

The contents described in this text are all verified on the platform of NVIDA's android2.2 and android4.0 system, Freescale's android2.3 system, Samsung's android4.0 system. At the present time, since there are a lot of the Android platforms, and the version of android is continuously updated, the source code has some differences on different platforms and versions. So we can not guarantee that the methods described in this text on all platforms and versions are validity. And the problems mentioned in the text, platform vendors may also have been modified. Therefore, do modify the system according to the code of the used platform in conjunction with this document.

This document includes two parts about the WCDMA branch and CDMA branch. Please adapt the system to the modem you used on you own according to the required function. If the patch is not needed, do not patch in order to avoid introducing other problems.

## 1 Inquire signal strength

**Question:** The system will not query the signal strength autonomously. (It's suitable for CDMA and WCDMA).

**Analysis:** As query signal strength is slow in the system, we use the Unsolicited reported method in RIL. This is in the purpose of displaying the signal strength as soon as the modem is up. But this will cause the system no longer querying the signal strength. The reason is as follows:

1 After the SIM card is READY, the function queueNextSignalStrengthPoll() is called.

2 In the function of queueNextSignalStrengthPoll(), the variable dontPollSignalStrength is judged. If it is false, the function continues to implement and sends the message of EVENT_POLL_SIGNAL_STRENGTH in delay.

In the function of handleMessage in the CdmaServiceStateTracker.java file, when receive the message of EVENT_POLL_SIGNAL_STRENGTH, the function getSignalStrength is called in the purpose of query singal strength, and then is converted to REQUEST to send down to RIL.

When the query is completed there will be an event reported up, then call the function queueNextSignalStrengthPoll() again. This formed a cycle of query.

If we use the Unsolicited reported method in the RIL, there will be the message about EVENT_SIGNAL_STRENGTH_UPDATE, and the value of dontPollSignalStrength will be true, thus the Unsolicited report will stop in the function of queueNextSignalStrengthPoll().

**Solution:** If we use the Unsolicited reported method of signal strength in the RIL, and with the hope that the system is still normal to inquire the signal strength, we could revise the function of queueNextSignalStrengthPoll to not judge the value of dontPollSignalStrength.

Modify the file as follows:

Frameworks/base/telephony/Java/com/android/internal/telephony/gsm/GsmServiceStateTracker.java

```
            private void queueNextSignalStrengthPoll() {
                - if (dontPollSignalStrength || (cm.getRadioState().isCdma())) {
                +   if   (cm.getRadioState().isCdma()) {
                    // The radio is telling us about signal strength changes
                       // we don't have to ask it
                       return;
                  }
```

# 2 Sending and receiveing MMS

## 2.1 The file needed to modify

Modify the file named GsmDataConnection.java as follows in the directory of ../frameworks/base/telephony/java/com/android/internal/telephony/gsm/:

```
 import android.os.SystemClock;
+import android.os. SystemProperties;
import android.util.Config;
……

Protected void onConnect(ConnectionParsms cp) {
apn = cp.apn;
+String oldapntype;
          if (DBG) log("Connecting to carrier: '" + apn.carrier
                    + "' APN: '" + apn.apn
                    + "' proxy: '" + apn.proxy + "' port: '" + apn.port);
          setHttpProxy (apn.proxy, apn.port);

+oldapntype = SystemProperties.get("ril. apn.type");
+log("old apn type ="+oldapntype);
+log("new apn type ="+apn.types[0]);
+if(apn.types[0].equals(Phone.APN_TYPE_MMS))
+{
+    SystemProperties.set("ril. apn.type", Phone.APN_TYPE_MMS);
+}
+else
+{
+     SystemProperties.set("ril. apn.type", Phone.APN_TYPE_DEFAULT);
+}

createTime = -1;
lastFailTime = -1;
lastFailCause = FailCause.NONE;
……
```

## 2.2 The APN settings

If you want to send MMS, you need to set two APNs. The first is used to access the Internet, and the second is used to send MMS. The APN used to access the Internet must be set as default, and the APN used to send MMS must be set as mms. If it is not set as the above, the function of mms can’t be work properly.

The setting of APN to send MMS under China Unicom refer to the table2.2.1 below (The specific details you can consult to the local Unicom operators):

Table 2.2.1 The APN settings

| | APN for Internet | APN for MMS |
|---|---|---|
| Name | default | mms |
| APN | 3gnet | 3gwap |
| MMSC | / | http://mmsc.myuni.com.cn |
| MMS Proxy | / | 10.0.0.172 |
| MMS Port | / | 80 |
| MCC | 460 | 460 |
| MNC | 01 | 01 |
| APN Type | default | mms |

# 3 The operation of SIM card

## 3.1 Save the Chinese contacts in the SIM card

In the existing android OS’s phonebook storage process, when you save your contacts which are write in Chinese to the SIM card, the Chinese parts will be lost. The reason is that in the Android OS there is no Chinese encoding process. Therefore, it will result in great inconvenience. If you want to add the function of saving the Chinese contacts in the SIM card, you need to do some modifications in the layer of framework to add Chinese encoding process. The file which you need to modify is as follows：

…\frameworks\base\telephony\java\com\android\internal\telephony\AndRecord.java.

The main changes are in the function named “public byte[] buildAdnString(int recordSize)”， and the modifications are as follows:

```
@@ -191,7 +199,9 @@
          admString[footerOffset + AND_EXTENSION_ID]
                    = (byte) 0xFF;
+         byte[] byteTagTemp = new byte[15];
+         if (alphaTag.getBytes().length != alphaTag.length()) {//Including Chinese
+                    if(alphaTag.length() >=7)
+                     {
+                       Log.w(LOG_TAG, “[buildAdnstring] Max length of   Chanese tag is 6”);
+                          return null;
+                     }
+                     try {
+                          byteTag = alphaTag.getBytes("utf-16BE");
+                           for(int i = 0; i < byteTag.length; i++)
+                           {
+                                byteTagTemp[i+1] = (byte) (byteTag[i] & 0xff);
+                           }
+                           for(int j = byteTag.length + 1; j < byteTagTemp.length; j++)
+                           {
+                                 byteTagTemp[j] = (byte)0xff;
+                           }
+                           byteTagTemp[0] = (byte)0x80;
+                    System.arraycopy(byteTagTemp, 0, adnString, 0, byteTag.length + 1);
+               }
+               catch (java.io.UnsupportedEncodingException ex) {
+                     Log.e("AdnRecord", "alphaTag convert byte exception");
+               }
+                else if (!TextUtils.isEmpty(alphaTag))
+               {//Not Including Chinese
                  //if (!TextUtils.isEmpty(alphaTag))
                    byteTag = GsmAlphabet.stringToGsm8BitPacked(alphaTag);
                    System.arraycopy(byteTag, 0, adnString, 0, byteTag.length);
               }
           return adnString;
          }
 }
```

If the system has been compiled, you only need to modify and recompile the files in the layer of framework. Such modifications are applicable to modification for the OS of android2.2, android2.3 and android4.0. We have already compiled and debugged this function successfully in the OS systems.

# 4 Voice services

## 4.1 Configuring the voice channel

**Question:** The modem of our company provides several modes of voice channel, different customers can configure according to their own needs.

**Analysis:** In the provided RIL library retains the interface, the customers need to set the property as follows in order to inform the RIL that which voice channel is used currently.

**Solution:** Modify the file named ril.c under the directory of /hardware/ril/rild/, which informs the RIL that the voice channel is used currently by the way to set the property in the system.

```
--- init/rild/rild.c      2011-10-31 14:28:40.054173000 +0800
+++ modified/rild/rild.c  2011-11-21 11:15:07.132922000 +0800
@@ -41,6 +41,13 @@
+#define ZTE_AUDIO_SWITCH "ril.audio.switch"
+#define ZTE_AUDIO_PCM "0"
+#define ZTE_AUDIO_LINEIN_LINEOUT_DIFF "1"
+#define ZTE_AUDIO_MIC_LINEOUT_LEFT_RIGHT "2"
+#define ZTE_AUDIO_MIC_LINEOUT_DIFF "3"
static void usage(const char *argv0)
 {
        fprintf(stderr, "Usage: %s -l <ril impl library> [-- <args for impl library>]\n", argv0);
@@ -127,6 +134,7 @@
+       property_set(ZTE_AUDIO_SWITCH, ZTE_AUDIO_MIC_LINEOUT_DIFF, NULL);

        if (rilLibPath == NULL) {
            if ( 0 == property_get(LIB_PATH_PROPERTY, libPath, NULL)) {
```

# 5 Data services

## 5.1 Customize the dail-up number for data

**Question:** In usual, the dail-up number for data is *99# in WCDMA and #777 in CDMA. But in some cases, the customers need to customize dail-up number. Therefore, some modifications have to be done as in this section.

**Analysis:** In the provided RIL library retains the interface. The customers need to call the REQUEST of RIL_REQUEST_SETUP_DATA_CALL from the upper layer, and in this REQUEST the customer must write the dail-up number to the array of DATA. At the same time, the customers need to set the property as follows in order to inform which parameter is the dial-up number in the array of DATA.

**Solution:** Modify the upper layer, send the dail-up number to the parameter of data in RIL_REQUEST_SETUP_DATA_CALL, and modify the paremeter of datalen. You can refer to commented code in the file named ril.h, as follows:

```
/**
* RIL_REQUEST_SETUP_DATA_CALL
*
* Setup a packet data connection
*
* "data" is a const char **
* ((const char **)data)[0] indicates whether to setup connection on radio technology CDMA or
  GSM/UMTS, 0-1. 0 - CDMA, 1-GSM/UMTS
*
* ((const char **)data)[1] is a RIL_DataProfile (support is optional)
* ((const char **)data)[2] is the APN to connect to if radio technology is GSM/UMTS. This APN
* will override the one in the profile. NULL indicates no APN overrride.
* ((const char **)data)[3] is the username for APN, or NULL
* ((const char **)data)[4] is the password for APN, or NULL
* ((const char **)data)[5] is the PAP / CHAP auth type. Values:
*                         0 => PAP and CHAP is never performed.
*                         1 => PAP may be performed; CHAP is never performed.
*                         2 => CHAP may be performed; PAP is never performed.
*                         3 => PAP / CHAP may be performed - baseband dependent.
```

Using the code below, you can read the defined dial-up number from the incoming data in RIL library.

*phonenumber = ((const char **)data)[phonenumber_position_int];*

The *phonenumber_position_init* informs us which parameter is the dial-up number in the array the data point to. And this paremeter needs to be written into the property. In the RIL library, the property value of *ril.phonenumber.postion* will be read to determine which paremeter is the dial-up number. The file named rild.c is locate in the directory of /hardware/ril/rild/, now we provide an implementation.

```
---     init/rild.c
+++ modified/rild.c
@@-38,6 +38,9@@
#define LIB_ARGS_PROPERTY     "rild.libargs"
#define MAX_LIB_ARGS          16
```

```
+ #define NUMBER_POSITION_VENDOR "06"
+ #define ZTE_PHONENUMBER_POSITON "ril.phonenumber.postion"

static void usage(const char *argv0)
{
        fprintf(stderr, "Usage: %s -l <ril impl library> [-- <args for impl library>]\n",
argv0);
        exit(-1);
}

@@-122,6 +124,10 @@
                }
            }
+ poperty_set(ZTE_PHONENUMBER_POSITON, NUMBER_POSITION_VENDOR);
if (rilLibPath == NULL) {
                if ( 0 == property_get(LIB_PATH_PROPERTY, libPath, NULL)) {
                    // No lib sepcified on the command line, and nothing set in props.
```

"#define NUMBER_POSITION_VENDOR "06"" indicate that the dial-up number is stored in ((const char **)data)[6], customer can set it according to their needs. Noting that in the macro definition, please use two digits to represent NUMBER_POSITION_VENDOR.

## 6 AT commands sending in the layer of Framework

In the provided RIL retains the interface to send AT command from framework. The users can call the interface function to achieve the request sending and response receiving in the layer of framework. AT commands are divided into several kinds, such as singleline, no_result, numeric, multiline and multiline_no_prefix. If the AT command is singleline or multiline, the paremeter *((char **)data)[2]* must be set; if the AT command is failed to send, an error will be returned, or success will be returned; if the AT command is no_result, there will be no response to be returned. The realization in RIL is as follows:

```
void requestOemHookStrings(void *data, size_t datalen, RIL_Token t)
{
        int err = 0;
        char * category = ((char **)data)[0];
        char *cmd = ((char **)data)[1];
        char *p_prefix = NULL;
        char **p_response_strings = NULL;
        ATResponse *p_response = NULL;
        int countLines=0;
        int count;
        ATLine *p_cur;
        if(strcmp(category, "singleline") == 0)
        {
            p_prefix = ((char **)data)[2];
            err = at_send_command_singleline(cmd, p_prefix, &p_response);
        }
        else if (strcmp(category, "no_result") == 0)
        {
            err = at_send_command(cmd, &p_response);
        }
        else if (strcmp(category, "numeric") == 0)
        {
            err = at_send_command_numeric(cmd, &p_response);
        }
        else if (strcmp(category, "multiline") == 0)
        {
            p_prefix = ((char **)data)[2];
            err = at_send_command_multiline(cmd, p_prefix, &p_response);
        }
        else if (strcmp(category, "multiline_no_prefix") == 0)
        {
            err = at_send_command_multiline_no_prefix(cmd, &p_response);
        }
        else
        {
            LOGE("ERROR:   requestOEMHookString:   unknown   category   \"%s\"\n",
    category);
            RIL_onRequestComplete(t, RIL_E_GENERIC_FAILURE, NULL, 0);
            return;
        }
        if (err < 0 || p_response->success == 0) goto error;
        /* count numbers of lines */
        for (countLines = 0, p_cur = p_response->p_intermediates
                ; p_cur != NULL
                ; p_cur = p_cur->p_next
        ) {
            countLines++;
       LOGD("%d", countLines);
       LOGD("p_cur = %s",p_cur->line);
        }
        if (countLines > 0){
            p_response_strings = malloc(sizeof(char*) * countLines);
            p_cur = p_response->p_intermediates;
            for (count= 0; count < countLines; count++)
            {
```

```
                        p_response_strings[count] = p_cur->line;
                        p_cur = p_cur->p_next;
                    }
        RIL_onRequestComplete(t, RIL_E_SUCCESS, &p_response_strings[0], sizeof(char*) *
        count);
                    free(p_response_strings);
                }
                else {
                    RIL_onRequestComplete(t, RIL_E_SUCCESS, NULL, 0);
                }
                at_response_free(p_response);
                return;
                error:
                if(p_response!=NULL)
                {
                    at_response_free(p_response);
                }
                LOGE("ERROR: requestOEMHookString() failed\n");
                RIL_onRequestComplete(t, RIL_E_GENERIC_FAILURE, NULL, 0);
                return;
            }
```

Now we will illustrate by an example to explain the method to send AT command for setting voice calls in upper layer after receivint the message of SIM_READY. In the function of onSimReady(), the function named sendAtCmd is called, and in sendAtCmd the system interface of *phone.mCM.invokeOemRilRequestStrings* is called too. Then the message is transformed to a REQUEST, and the message of EVENT_GET_OEM_RESPONSE which includes the response of AT command in this modem will be received in the function of handleMessage.

The file named SIMRecords.java in the directory of /frameworks/base/telephony/java/com/android/internal/telephony/gsm/ is modified as follows:

```
--- init/SIMRecords/SIMRecords.java   2011-10-31 14:29:11.364172000 +0800
+++ modified/SIMRecords/SIMRecords.java      2011-12-28 08:56:20.014181000 +0800
@@ -140,6 +140,8 @@
        private static final int EVENT_SET_MSISDN_DONE = 30;
        private static final int EVENT_SIM_REFRESH = 31;
        private static final int EVENT_GET_CFIS_DONE = 32;
+        private static final int EVENT_GET_OEM_RESPONSE = 33;
        // ***** Constructor
@@ -464,6 +466,7 @@
        AdnRecord adn;
        byte data[];
+       String AT_response[];
        boolean isRecordLoadResponse = false;

@@ -475,7 +478,16 @@
```

```
        case EVENT_RADIO_OFF_OR_NOT_AVAILABLE:
           onRadioOffOrNotAvailable();
           break;
+       case EVENT_GET_OEM_RESPONSE:
+          ar = (AsyncResult)msg.obj;
+          if(ar.exception != null)
+          {
+               Log.e(LOG_TAG, "Exception send oem AT:" + ar.exception);
+          }
+          AT_response = (String[])(((AsyncResult)msg.obj).result);
+          Log.d(LOG_TAG, "[AT]:" + AT_response[0]);
+          break;
        /* IO events */
        case EVENT_GET_IMSI_DONE:
           isRecordLoadResponse = true;
@@ -1184,8 +1196,19 @@
        SimCard.INTENT_VALUE_ICC_READY, null);
              fetchSimRecords();
+       sendAtCmd();
        }
+       private void sendAtCmd()
+       {
+             String[] cmd1={"no_result","AT+ZVOCCH=1"};
+             String[] cmd2={"no_result", "AT+ZAUDIO_HUNGUP_DELAY=2000"};
+             phone.mCM.invokeOemRilRequestStrings(cmd1,
      obtainMessage(EVENT_GET_OEM_RESPONSE));
+             phone.mCM.invokeOemRilRequestStrings(cmd2,
      obtainMessage(EVENT_GET_OEM_RESPONSE));
+       }
        private void fetchSimRecords() {
              recordsRequested = true;
              IccFileHandler iccFh = phone.getIccFileHandler()
```

# Part III： Modify the android system for CDMA modem

## Special Explanation

The contents described in this text are all verified on the platform of NVIDA’s android2.2 and android4.0 system, Freescale’s android2.3 system, Samsung’s android4.0 system. At the present time, since there are a lot of the Android platforms, and the version of android is continuously updated, the source code has some differences on different platforms and versions. So we can not guarantee that the methods described in the text on all platforms and versions are effective. And the problems mentioned in the text, platform vendors may also have been modified. Therefore, do modify the system according to the code of the platform used in conjunction with this document.

This document includes two parts of the WCDMA branch and CDMA branch. Please adapt your system to your modem by yourself according to the modems you used and the functionality you required. Do not patch if the patch is not needed in order to avoid introducing other problems.

# 1 The switch method between CDMA and GSM branches

In the file named RILConstants.java under the directory of /frameworks/base/telephony /java/com/android/internal/telephony, the acquiescent type of phone at the time of first started is defined. And it is defined by the parameter named PREFERRED_NETWORK_MODE in the struct of RILConstants. We recommend that if the customer uses the modem of WCDMA, the value of this parameter must be set as NETWORK_MODE_WCDMA_PREF, and if uses CDMA modem it need to be set as NETWORK_MODE_CDMA. The methods that only replace the compiled file of framework.jar may take no effect, as it will cause the unstability of OS after starting up. The right way is re-programming the system and loading the image after changed.

```
public interface RILConstants {
……
        int NETWORK_MODE_WCDMA_PREF          = 0; /* GSM/WCDMA (WCDMA preferred) */
        int NETWORK_MODE_GSM_ONLY            = 1; /* GSM only */
        int NETWORK_MODE_WCDMA_ONLY          = 2; /* WCDMA only */
        int NETWORK_MODE_GSM_UMTS            = 3; /* GSM/WCDMA (auto mode, according
to PRL)
                                              AVAILABLE Application Settings menu*/
        int NETWORK_MODE_CDMA                = 4; /* CDMA and EvDo (auto mode,
according to PRL)
                                              AVAILABLE Application Settings menu*/
        int NETWORK_MODE_CDMA_NO_EVDO        = 5; /* CDMA only */
        int NETWORK_MODE_EVDO_NO_CDMA        = 6; /* EvDo only */
        int NETWORK_MODE_GLOBAL              = 7; /* GSM/WCDMA, CDMA, and EvDo (auto
    mode, according to PRL)      AVAILABLE Application Settings menu*/
-       int PREFERRED_NETWORK_MODE           = NETWORK_MODE_WCDMA_PREF;
+       int PREFERRED_NETWORK_MODE           = NETWORK_MODE_CDMA;
        /* CDMA subscription source. See ril.h RIL_REQUEST_CDMA_SET_SUBSCRIPTION */
        int SUBSCRIPTION_FROM_RUIM           = 0; /* CDMA subscription from RUIM when
available */
        int SUBSCRIPTION_FROM_NV             = 1; /* CDMA subscription from NV *
        ……
        }
```

If there is an Error of “phone process terminated unexpectedly” after you did all modifications, built the system and downloaded the image as above correctly, you should firstly check whether the library of libreference-ril.so is pushed into the directory of /system/lib. If you have pushed, the Error is still there, you can try as follows to modify the file and then build and download the system.

Modify the function of makeDefaultPhone() in the file named PhoneFactory.java in the directory of /frameworks/base/telephony/java/com/android/internal/telephony/ as follows.

```
        Log.i(LOG_TAG,”Network Mode set to” + Integer.toString(networkMode));
+       networkMode = 4;
        // Get cdmaSubscription
```

# 2 SMS related issues

## 2.1 SMS delivery reports

**Question:** When the phone is set a delivery report for each message you send in CDMA, it will receive the SMS receipt about “!” if you send the SMS success.

**Analysis:** This phenomenon illustrates that the status of SMS is not correctly parsed in the upper UI layer. Please check the code at the function of getStatus in the file named SmsMessage.java located in the directory of /frameworks/base/telephony/java/com/android/internal/telephony/cdma/. If the code of this function is as follows, the reason why the user can’t read the state report may be that the SMS delivery reports is stored in the previous of 16 bit at the framework layer, but the UI layer is matched with GSM which stored in the next 16 bit.

```
（447）public int getStatus() {
                return (status << 16);
                }
```

**Solution:** Modify the function as follows.:

```
（447）public int getStatus() {
                return status;
                }
```

## 2.2 The sending of special characters in SMS

**Question:** When sending special characters such as “`”, the terminal equipment will receive “?” or hash code. But the function of sending this SMS to itself is normal.

**Analysis:** In the android OS, if the contents of SMS are all characters, the encoding of CDMA is 7bit-ASCII. At present, some terminal equipment can’t handle parts of special characters in 7bit-ASCII encoding correctly. Therefore, this will result in receiving abnormal. The other encoding format is UNICODE which don’t have the problem said above, but the defect is that it will occupy 16 bits. To solve this problem, we offer a solution that we will use the encoding of

UNICODE if the existence of these special characters is found.

**Solution:** (Such modifications are applicable to modification for CDMA as well as WCDMA**):**

```
--- init/BearerData.java   2010-12-11 05:13:42.000000000 +0800
+++ modified/BearerData.java       2011-08-31 14:51:03.602914000 +0800
@@ -395,6 +395,12 @@
            int msgLen = msg.length();
            if (force) return msgLen;
            for (int i = 0; i < msgLen; i++) {
+           if(UserData.charToAscii.get(msg.charAt(i), -1) == 96)
+           {
+               Log.d(LOG_TAG, "[ZTE]encode find '`', return 2");
+               return -2;
+           }
            if (UserData.charToAscii.get(msg.charAt(i), -1) == -1)
            {
                return -1;
            }
@@ -410,9 +416,10 @@
        */
      public static TextEncodingDetails calcTextEncodingDetails(CharSequence msg,
                 boolean force7BitEncoding)
        {
            TextEncodingDetails ted;
-           int septets = countAsciiSeptets(msg, force7BitEncoding);
-           if (septets != -1 && septets <= SmsMessage.MAX_USER_DATA_SEPTETS) {
+            int septets = countAsciiSeptets(msg, force7BitEncoding);
+            if (septets != -1 && septets <= SmsMessage.MAX_USER_DATA_SEPTETS
                 && septets != -2) {
                 ted = new TextEncodingDetails();
                 ted.msgCount = 1;
                 ted.codeUnitCount = septets;
@@ -421,7 +428,7 @@
            } else {
                 ted = com.android.internal.telephony.gsm.SmsMessage.calculateLength(
                           msg, force7BitEncoding);
-                if (ted.msgCount == 1 && ted.codeUnitSize == SmsMessage.ENCODING_7BIT)
                 {

+                     if ((ted.msgCount == 1
                     && ted.codeUnitSize == SmsMessage.ENCODING_7BIT)||septets == -2) {
                     // We don't support single-segment EMS, so calculate for 16-bit
                     // TODO: Consider supporting single-segment EMS
                     ted.codeUnitCount = msg.length();
@@ -448,6 +455,12 @@
```

```
                    int msgLen = msg.length();
                    for (int i = 0; i < msgLen; i++) {
                          int charCode = UserData.charToAscii.get(msg.charAt(i), -1);
+                   if(charCode == 96)
+                   {
+                        Log.d(LOG_TAG, "[ZTE]encode find '`', use unicode encode");
+                        charCode = -1;
+                   }
                    if (charCode == -1) {
                         if (force) {
                                        outStream.write(7, UserData.UNENCODABLE_7_BIT_CHAR);
```

NOTE: During the test, we only found that the character of "`" can't be sent. If there are other characters which have the same phenomenon, you could do some modifications according to this method.

## 2.3 The receiption of cascade SMS written in English

**Question:** There is parsing error when receive the cascade SMS written in English.

**Analysis:** In CDMA, the received cascade SMS which is written in English is generally encoded by 7bit-ASCII. Compared to single SMS, it adds the parsing of the SMS header. At the same time, it skips the corresponding stuffing bits.

**Solution:** Modify the function of decode7bitAscii() in the file named BearerData.java at the directory of /frameworks/base/telephony/java/com/android/internal/telephony/cdma/sms/. The modification is as follows.

```
--- init/BearerData.java   2010-12-11 05:13:42.000000000 +0800
+++ modified/BearerData.java      2011-08-31 14:51:03.602914000 +0800
      @@ -924,20 +1144,35 @@
                    }
              }
              private static String decode7bitAscii(byte[] data, int offset, int numFields)
                    throws CodingException
                    try {
      -                    offset *=    8;
      +                    int offsetBits = offset * 8;
                           StringBuffer strBuf = new StringBuffer(numFields);
                           BitwiseInputStream inStream = new BitwiseInputStream(data);
      -                    int wantedBits = (offset * 8) + (numFields * 7);
      +                     int wantedBits =    numFields * 7;
                           if (inStream.available() < wantedBits) {
                                 throw new CodingException("insufficient data (wanted " + wantedBit "
                                 " + inStream.available() + ")");
                           }
```

```
-                  inStream.skip(offset);
+           if(offsetBits%7 != 0)
+           {
+                  int fillbit = 7-offsetBits%7;
+                  inStream.skip(offsetBits);
+                  inStream.skip(fillbit); /*fill bit*/
+                  offset++;
+           }
+           else
+           {
+                  inStream.skip(offsetBits);
+           }
            for (int i = 0; i < (numFields-offset); i++) {
                   int charCode = inStream.read(7);
                   if ((charCode >= UserData.ASCII_MAP_BASE_INDEX) &&
                       (charCode <= UserData.ASCII_MAP_MAX_INDEX)) {
```

## 2.4 Using the UNICODE to send the casecade SMS written in English

**Question:** At present, the modem doesn't support the sending of casecade English SMS encoded in 7bit-ASCII.

**Analysis:** At present, we solve this problem in the way of encoding in UNICODE temporarily.

**Solution:** Modify the code as follows.

Caculate the length of each SMS and the number of messages how many it is devided into when encoded in UNICODE.

```
---init/frameworks/base/telephony/java/com/android/internal/telephony/cdma/sms/BearerData.java
      2011-10-17 08:17:27.000000000 -0400
+++modify/frameworks/base/telephony/java/com/android/internal/telephony/cdma/sms/BearerDat
a.java      2011-10-17 08:04:28.000000000 -0400
@@ -412,6 +412,12 @@
              boolean force7BitEncoding) {
          TextEncodingDetails ted;
          int septets = countAsciiSeptets(msg, force7BitEncoding);
+         int islongenglishsms = 0;
+         if(septets > SmsMessage.MAX_USER_DATA_SEPTETS)
+         {
+                islongenglishsms = 1;
+                Log.d(LOG_TAG, "this is a long english sms");
+         }
          if (septets != -1 && septets <= SmsMessage.MAX_USER_DATA_SEPTETS) {
              ted = new TextEncodingDetails();
              ted.msgCount = 1;
@@ -421,7 +427,7 @@
```

```
                } else {
                        ted = com.android.internal.telephony.gsm.SmsMessage.calculateLength(
                                        msg, force7BitEncoding);
-                       if (ted.msgCount == 1 && ted.codeUnitSize == SmsMessage.ENCODING_7BIT)
{
+                                          if ((ted.msgCount == 1 && ted.codeUnitSize ==
SmsMessage.ENCODING_7BIT)||islongenglishsms == 1) {
                                // We don't support single-segment EMS, so calculate for 16-bit
                                // TODO: Consider supporting single-segment EMS
                                ted.codeUnitCount = msg.length();
```

In the version of android2.2 and android2.3 OS, if there is a long casecade SMS, it will be encoded in UNICODE. Modify the function of sendMultipartText() in the file named CdmaSMSDispatcher.java as follows.

```
---init/frameworks/base/telephony/java/com/android/internal/telephony/cdma
/CdmaSMSDispatcher.java     2011-10-17 08:23:23.000000000 -0400
+++modify/frameworks/base/telephony/java/com/android/internal/telephony/cdma
/CdmaSMSDispatcher.java     2011-10-17 08:23:41.000000000 -0400
@@ -407,6 +407,12 @@
                        } else { // assume UTF-16
                                uData.msgEncoding = UserData.ENCODING_UNICODE_16;
                        }
+                        if(msgCount>1)
+                        {
+                               uData.msgEncoding = UserData.ENCODING_UNICODE_16;
+                        }
                        uData.msgEncodingSet = true;
                        /* By setting the statusReportRequested bit only for the
```

In the version of android4.0 OS, if there is a long casecade SMS, it will be encoded in UNICODE too. But there are some differences between the source code of android4.0, android2.2 and android 2.3. Therefore, the modification is different. You have to modify the three files named SMSDispathcer.java, CdmaSMSDispathcer.java and GsmSMSDispathcer.java.

---init/frameworks/base/telephony/java/com/android/internal/telephony/SMSDispathcer.java

In the function of sendMultiPartText(), Modify as follows:

```
-               sendNewSubmitPdu(destAddr, scAddr, parts.get(i), smHeader, encoding,
-                       sentIntent, deliveryIntent, (I == (msgCount -1)));
+               sendMutiSubmitPdu(destAddr, scAddr, parts.get(i), smHeader, encoding,
+                   sentIntent, deliveryIntent, (I == (msgCount -1)));
        }
}
/**
  *Create a new SubmitPdu and send it
  */
```

```
+   protected abstract void sendMutiSubmitPdu(String destinationAddress, String scAddress,
+            String message, SmsHeader smsHeader, int encoding,
+                PendingIntent sentIntent, PendingIntent deliveryIntent, Boolean lastPart, int
msgCount);

protected abstract void sendNewSubmitPdu(String destinationAddress, String scAddress,
                String message, SmsHeader smsHeader, int encoding,
                PendingIntent sentIntent, PendingIntent deliveryIntent, Boolean lastPart);
```

Modify the file:
---init/frameworks/base/telephony/java/com/android/internal/telephony/cdma/CdmaSMSDispathcer.java

In this file, add the function sendMutiSubmitPdu() below the function of sendNewSubmitPdu().

```
+   protected abstract void sendMutiSubmitPdu(String destinationAddress, String scAddress,
+       String message, SmsHeader smsHeader, int encoding,
+       PendingIntent sentIntent, PendingIntent deliveryIntent, Boolean lastPart, int msgCount)   {
+           UserData uData = new UserData();
+           uData.payloadStr = message;
+           uData.userDataHeader = smsHeader;
+           if (encoding == android.telephony.SmsMessage.ENCODING_7BIT)   {
+               uData.msgEncoding = UserData.ENCODING_GSM_7BIT_ALPHABET;
+           } else { //assume UTF-16
+               uData.msgEncoding = UserData.ENCODING_UNICODE_16;
+           }
+           if (msgCount > 1)
+           {
+               uData.msgEncoding = UserData.ENCODING_UNICODE_16;
+           }
+           uData.msgEncodingSet = true;
+
+           /* By setting the statusReportRquested bit only for thr
+             * last message fragment, this will result in only one
+             * callback to the sender when that last fragment delivery
+             * has been acknowledge. */
+             SmsMessage.SubmitPdu submitPdu = SmsMessage.getSubmitPdu(destinationAddress,
+                   uData, (deliveryInten !=null) && lastPart);
+
+             sendSubmitPdu(submitPdu, sentIntent, deliveryInten);
```

Modify the file:
---init/frameworks/base/telephony/java/com/android/internal/telephony/gsm/GsmSMSDispathcer.java

In this file, add the function sendMutiSubmitPdu() below the function of sendNewSubmitPdu().

```
+   protected abstract void sendMutiSubmitPdu(String destinationAddress, String scAddress,
+       String message, SmsHeader smsHeader, int encoding,
+       PendingIntent sentIntent, PendingIntent deliveryIntent, Boolean lastPart, int msgCount)    {
+     SmsMessage.SubmitPdu pdu = SmsMessage.getSubmitPdu(scAddress. destimationAddress,
+           message, deliveryIntent != null, SmsHeader,toByteArray(smsHeader),
+           encoding, smsHeader,languageTable, smsHeader.languageShiftTable);
+       if(pdu != null)    {
+           sendRawPdu(pdu.encodedScAddress, pdu.encodedMessage, sendIntent, deliveryIntent);
+       }    else  {
+         Log.e(TAG, "GsmSMSDispatcher.sendNewSubmitPdu(): getSubmitPdu() returned null");
+       }
+   }
```

# 3 The function of STK

## 3.1 Applicable conditions

There is no correlate processing about the part of STK in some source code of the Android2.2 and Android2.3 platforms. The correlate processing about the STK in GSMPhone is at the package of com.android.internal.telephony.gsm.stk. As there are many differences of class member variables and methods between the CDMAPhone and GSMPhone, we can't use the STK service in GSM directly. We need to do some modifications in the layer of framework and cooperate with the UI layer to open the STK service in CDMA.

The other service to process the STK in the layer of framework is Catservice, which can be used in both CDMA and GSM. This part will introduce the way how to transplant the Catservice to the branch of CDMA.

The source code of android4.0 has already included the process of STK. The customers don't need to modify themselves.

## 3.2 Download the code

The code of Catservice in the layer of framework can be downloaded at the following address:

http://hi-android.info/src/com/android/internal/telephony/cat/

The download address of correspongding STK application using Catservice is as follows:

http://hi-android.info/src/com/android/stk/

## 3.3 The modification

Put the downloaded folder named Cat into the directory of /frameworks/base/telephony/java/com/android/internal/telephony.

Put the downloaded folder named Stk into the directory of /android2.2/packages/apps.

The file needed to modified：
Frameworks/base/telephony/java/com/android/internal/telephony/CDMAPhone.java
1 Add: (57) *import com.android.internal.telephony.cat.CatService;*
2 Add: add the member variable in the class of CDMAPhone:
(112)*CatService mCcatService*
3Add: add the code as follows in the constructor of the class of CDMAPhone:
(167) *mCcatService = CatService.getInstance(mCM, mRuimRecords, mContext, mIccFileHandler, mRuimCard);*
Modify the file：
/frameworks/base/telephony/java/com/android/internal/telephony/baseCommands.java
1 Modify: （398）*setOnStkSessionEnd->setOnCatSessionEnd*
2 Modify: （402）*unSetOnStkSessionEnd->unSetOnCatSessionEnd*
3 Modify: （406）*setOnStkProactiveCmd->setOnCatProactiveCmd*
4 Modify: （410）*unSetOnStkProactiveCmd->unSetOnCatProactiveCmd*
5 Modify: （414）*setOnStkEvent->setOnCatEvent*
6 Modify: （418）*unSetOnStkEvent->unSetOnCatEvent*
7 Modify: （422）*setOnStkCallSetUp->setOnCatCallSetUp*
8 Modify: （426）*unSetOnStkCallSetUp->unSetOnCatCallSetUp*
Modify the file:
Framework/base/telephony/java/internal/telephony/CommandsInterface.java
1 Modify: （384）*setOnStkSessionEnd->setOnCatSessionEnd*
2 Modify: （385）*unSetOnStkSessionEnd->unSetOnCatSessionEnd*
3 Modify: （396）*setOnStkProactiveCmd->setOnCatProactiveCmd*
4 Modify: （397）*unSetOnStkProactiveCmd->unSetOnCatProactiveCmd*
5 Modify: （407）*setOnStkEvent->setOnCatEvent*
6 Modify: （408）*unSetOnStkEvent->unSetOnCatEvent*
7 Modify: （418）*setOnStkCallSetUp->setOnCatCallSetUp*
8 Modify: （419）*unSetOnStkCallSetUp->unSetOnCatCallSetUp*
Modify the file:
/framework/base/telephony/java/android/internal/telephony/gsm/stk/stkService.java
1 Modify: （163）*mCmdIf.setOnStkSessionEnd(this, MSG_ID_SESSION_END, null);）->*
*mCmdIf.setOnCatSessionEnd(this, MSG_ID_SESSION_END, null);）*
2 Modify: （164）*mCmdIf.setOnStkProactiveCmd(this, MSG_ID_PROACTIVE_COMMAND, null); ->*
*mCmdIf.setOnCatProactiveCmd(this, MSG_ID_PROACTIVE_COMMAND, null);*
3 Modify: （166）*mCmdIf.setOnStkEvent(this, MSG_ID_EVENT_NOTIFY, null);->*
*mCmdIf.setOnCatEvent(this, MSG_ID_EVENT_NOTIFY, null);*
4 Modify: （167）*mCmdIf.setOnStkCallSetUp(this, MSG_ID_CALL_SETUP, null);->*
*mCmdIf.setOnCatCallSetUp(this, MSG_ID_CALL_SETUP, null);*
5 Modify: （180）*mCmdIf.unSetOnStkSessionEnd(this);*
*mCmdIf.unSetOnStkProactiveCmd(this);*
*mCmdIf.unSetOnStkEvent(this);*
*mCmdIf.unSetOnStkCallSetUp(this);*

```
->
mCmdIf.unSetOnCatSessionEnd(this);
mCmdIf.unSetOnCatProactiveCmd(this);
mCmdIf.unSetOnCatEvent(this);
mCmdIf.unSetOnCatCallSetUp(this);
```

# 4 The sending and receiption of MMS

**Question: Can't send MMS.**

**Analysis:** The processes of sending MMS are as follows. Firstly, the user clicks the Send button. Secondly, the APN type of default for data connection will be disconnected at time of the MMS is detected in the OS. Thirdly, the APN type of mms for MMS is connected, and the content of this MMS is pushed to the internet. At last, the default APN for data connection is desterilized.

**Solution:** There is no function of APN setting in the branch of CDMA. Please refer to the section of 6 in the Part III to add the function of APN setting. The APN used to access the Internet must be set as default, and the APN used to send MMS must be set as mms. The setting of APN to send MMS under China Telecom refer to the table4.1 below (The specific details you can consult to the local Unicom operators):

Table 4.1 The settings of APN

| | APN for Internet | APN for MMS |
|---|---|---|
| Name | NET | MMS |
| APN | ctnet | ctwap |
| Username | card | ctwap@mycdma.cn |
| Password | card | vnet.mobi |
| MMSC | / | http://mmsc.vnet.mobi |
| MMS Proxy | / | 10.0.0.200 |
| MMS Port | / | 80 |
| MCC | 460 | 460 |
| MNC | 03 | 03 |
| APN Type | default | mms |

The type of APN is defined in the file named phone.java under the directory of Frameworks/base/telephony/java/com/android/internal/telephony. You can view the source code to see the specific details.

The problems will appear in the default source code in the process of receiving the MMS under China Telecom. That's because there are differences between the package type of Telecom and parsing in the upper layer of android. Telecom adds a layer of package. The medications are as follows:

1 Add a function of decodeUserData behind the function of “decodeUserData (BearerData bData, BitwiseInputStream inStream)” as follows in the file named BearerData.java located in the directory of telephony/java/com/android/internal/telephony/cdma/sms/. In this function, we add special treatment for the informing of MMS in Telecom.

```
private static boolean decodeUserData(BearerData bData, BitwiseInputStream inStream,
                                      int teleService)
throws BitwiseInputStream.AccessException
{
    int paramBits = inStream.read(8) * 8;
    Log.d(LOG_TAG, "decodeUserData paramBits: " + paramBits);
    bData.userData = new UserData();
    bData.userData.msgEncoding = inStream.read(5);
    Log.d(LOG_TAG, "decodeUserData msgEncoding: " + bData.userData.msgEncoding);
    bData.userData.msgEncodingSet = true;
    bData.userData.msgType = 0;
    int consumedBits = 5;
    if ((bData.userData.msgEncoding ==
        UserData.ENCODING_IS91_EXTENDED_PROTOCOL) ||
        (bData.userData.msgEncoding == UserData.ENCODING_GSM_DCS))
    {
        bData.userData.msgType = inStream.read(8);
        Log.d(LOG_TAG, "decodeUserData msgType: " + bData.userData.msgType);
        consumedBits += 8;
    }
    +if(SmsEnvelope.TELESERVICE_WAP_PUSH == teleService)
    +{
    +    bData.userData.numFields = inStream.read(8);
    +    consumedBits += 8;
    +    inStream.skip(69);
    +    consumedBits += 69;
    +    bData.userData.numFields -= 9;
    +    if(paramBits < consumedBits)
    +    {
    +        return false;
    +    }
    +     int dataBits = paramBits - consumedBits;
    +     bData.userData.payload = inStream.readByteArray(dataBits);
    + }
     else
     {
         bData.userData.numFields = inStream.read(8);
         Log.d(LOG_TAG, "decodeUserData numFields: " + bData.userData.numFields);
         consumedBits += 8;
         int dataBits = paramBits - consumedBits;
```

```
            Log.d(LOG_TAG, "decodeUserData dataBits: " + dataBits);
            bData.userData.payload = inStream.readByteArray(dataBits);
        }
        return true;
}
```

2 Add a function "decode(byte[] smsData, int teleService)"as follows behind the function "public static BearerData decode(byte[] smsData)" in the file named BearerData.java in the directory of telephony/java/com/android/internal/telephony/cdma/sms/.

```
public static BearerData decode(byte[] smsData, int teleService) {
        try {
            BitwiseInputStream inStream = new BitwiseInputStream(smsData);
            BearerData bData = new BearerData();
            int foundSubparamMask = 0;
            Log.d(LOG_TAG, "BearerData decode inStream.available() = " +
            inStream.available());
            while (inStream.available() > 0) {
                boolean decodeSuccess = false;
                int subparamId = inStream.read(8);
                int subparamIdBit = 1 << subparamId;
                if ((foundSubparamMask & subparamIdBit) != 0) {
                    throw new CodingException("illegal duplicate subparameter (" +
                                                      subparamId + ")");
                }

                Log.d(LOG_TAG, "BearerData decode subparamId = " + subparamId);
                switch (subparamId) {
                case SUBPARAM_MESSAGE_IDENTIFIER:
                    decodeSuccess = decodeMessageId(bData, inStream);
                    break;
                case SUBPARAM_USER_DATA:
+                   decodeSuccess = decodeUserData(bData, inStream, teleService);
                    break;
                case SUBPARAM_USER_REPONSE_CODE:
                    decodeSuccess = decodeUserResponseCode(bData, inStream);
                    break;
                ……
```

3 Add teleServiceId of TELESERVICE_WAP_PUSH in Telecom

```
      ---
    telephony_init/telephony/java/com/android/internal/telephony/cdma/sms/SmsEnvelope.java
       2010-12-22 15:31:30.000000000 -0500

      +++
telephony_modify/telephony/java/com/android/internal/telephony/cdma/sms/SmsEnvelope.
java    2011-09-12 02:14:43.000000000 -0400

    @@ -35,6 +35,8 @@

    static public final int TELESERVICE_VMN              = 0x1003;
    static public final int TELESERVICE_WAP               = 0x1004;
    static public final int TELESERVICE_WEMT              = 0x1005;
+   static public final int TELESERVICE_WAP_PUSH          = 0xFDEA;
    /**
     * The following are defined as extensions to the standard teleservices
```

4 Modify the file in the follow. Call the function defined above when parsing the MMS of Telecom.

```
---        telephony_init/telephony/java/com/android/internal/telephony/cdma/SmsMessage.java
    2010-12-22 15:31:30.000000000 -0500
+++    telephony_modify/telephony/java/com/android/internal/telephony/cdma/SmsMessage.java
    2011-09-23 05:30:12.000000000 -0400
@@ -543,7 +561,8 @@
                }
                return;
            }
-           mBearerData = BearerData.decode(mEnvelope.bearerData);
+            mBearerData = BearerData.decode(mEnvelope.bearerData, mEnvelope.teleService);
            if (Log.isLoggable(LOGGABLE_TAG, Log.VERBOSE)) {
                Log.d(LOG_TAG, "MT raw BearerData = '" +
                            HexDump.toHexString(mEnvelope.bearerData) + "'");
```

5 Judging if it’s a MMS according to teleServiceId, then reported to the upper layer to handle.

```
--- telephony_init/telephony/java/com/android/internal/telephony/cdma/CdmaSMSDispatcher.java
    2010-12-22 15:31:30.000000000 -0500
+++
telephony_modify/telephony/java/com/android/internal/telephony/cdma/CdmaSMSDispatcher.jav
a    2011-09-22 21:20:35.000000000 -0400
@@ -158,7 +158,9 @@
                return Intents.RESULT_SMS_OUT_OF_MEMORY;
            }

-           if (SmsEnvelope.TELESERVICE_WAP == teleService) {
+            if ( (SmsEnvelope.TELESERVICE_WAP == teleService) ||
+                   (SmsEnvelope.TELESERVICE_WAP_PUSH== teleService))
+   {
                return processCdmaWapPdu(sms.getUserData(), sms.messageRef,
                        sms.getOriginatingAddress());
            }
```

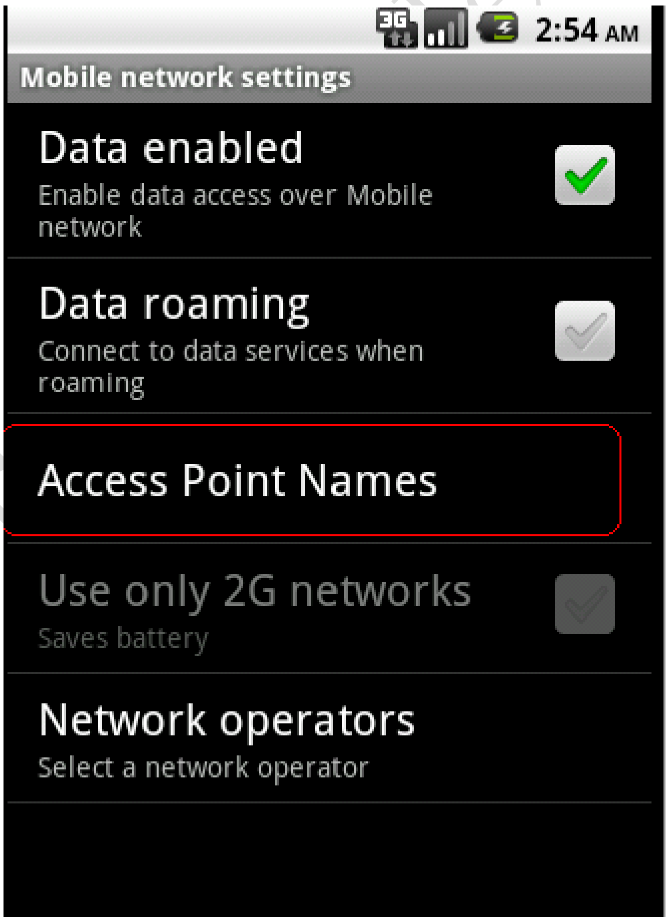
## 5 Inquire singnal strength

Please consult the chapter of 1 in part II

# 6 The modification instructions of APN Settings

## 6.1 Introduce

In the common Android source code of CDMA branch, there is no APN Setting part which should be located in the list menu of settings->Wireless&networks->Mobile networks in compiled system as the red line part in Figure 6.1.1 shows. Therefor there is no interface and menu after click "Access Point Names" button. In order to add this function we need to increase a lot of code in the layer of app and framework.



2:54 AM

Mobile network settings

Data enabled
Enable data access over Mobile network ✓

Data roaming
Connect to data services when roaming

Access Point Names

Use only 2G networks
Saves battery

Network operators
Select a network operator

Figure 6.1.1 APN function Interface

## 6.2 The modifications of related files in Android2.2 and 2.3

The modification mainly involves the following two parts:

1. In the layer of APP, the relevant file for displaying:

packages\apps\Phone\res\xml\cdma_options.xml

2. The related documents for internal processing

In the directory of frameworks\base\telephony\java\com\android\internal\telephony\cdma
CdmaDataConnection.java
CdmaDataConnectionTracker.java
RuimRecords.java
In the directory of frameworks\base\telephony\java\com\android\internal\telephony\gsm
ApnSetting.java

### 6.2.1 The modification of xml file to add APN function

#### 6.2.1.1 The file of cdma_options.xml

Add APN function in the file named cdma_options.xml

*xmlns:settings="http://schemas.android.com/apk/res/com.android.phone">*
*After this code, add the following content:*

```
<PreferenceScreen
        android:key="button_apn_key"
        android:title="@string/apn_settings"
        android:persistent="false">

        <intent android:action="android.intent.action.MAIN"
                android:targetPackage="com.android.settings"
                android:targetClass="com.android.settings.ApnSettings" />

</PreferenceScreen>
```

After adding the above code, it will display the red line part like Figure 6.1.1. When clicked that button, you can set the APN info.

### 6.2.2 The modification of java files

result1.TXT

#### 6.2.2.1 The modification of file named CdmaDataConnection.java

In the class of “public class CdmaDataConnection extends DataConnection”, there is the function named “protected void onConnect(ConnectionParams cp)”, which can set the APN as follows.

*phone.mCM.setupDataCall(Integer.toString(RILConstants.SETUP_DATA_TECH_CDMA),*

```
                              Integer.toString(dataProfile), null, null,
                              null,
         Integer.toString(RILConstants.SETUP_DATA_AUTH_PAP_CHAP), msg);
         Modify the code as follows:
         phone.mCM.setupDataCall(Integer.toString(RILConstants.SETUP_DATA_TECH_CDMA),
                              Integer.toString(dataProfile), cp.apn.apn, cp.apn.user,
                              cp.apn.password,
         Integer.toString(RILConstants.SETUP_DATA_AUTH_PAP_CHAP), (In the source code
         of Android2.3 we need to add RILConstants.SETUP_DATA_PROTOCOL_IP,) msg);
```

When call the RIL's "setupDataCall" to connect, it no longer use NULL, but use the APN we have already set.

### 6.2.2.2 CdmaDataConnectionTracker.java Modification

There are many modifications in this part, the functions we need to added are as follows.

```
private class ApnChangeObserver extends ContentObserver ()
private void createAllApnList()
private ArrayList<ApnSetting> createApnList(Cursor cursor)
private void setPreferredApn(int pos)
private String[] parseTypes(String types)
private String apnListToString (ArrayList<ApnSetting> apns)
private ApnSetting getNextApn()
private ApnSetting getPreferredApn()
private ArrayList<ApnSetting> buildWaitingApns()
private boolean IsPreferredApnChanged()
private void onApnChanged()
private ApnSetting getProjectionByApnId(String apnId)
private void setPppdApnFile(String key)
```

There are some other functions have been modified. The content is as follows.
Add "ApnChangeObserver" and other Variables.

```
//useful for debugging
 boolean failNextConnect = false;
+private class ApnChangeObserver extends ContentObserver {
+       public ApnChangeObserver () {
+              super(mDataConnectionTracker);
+       }
+
+       @Override
+       public void onChange(boolean selfChange) {
+              sendMessage(obtainMessage(EVENT_APN_CHANGED));
+       }
+}
+/**
```

```
+ * allApns holds all apns for this sim spn, retrieved from
+ * the Carrier DB.
+ *
+ * Create once after simcard info is loaded
+ */
+private ArrayList<ApnSetting> allApns = null;
+
+private ApnChangeObserver apnObserver;
+
+/**
+ * waitingApns holds all apns that are waiting to be connected
+ *
+ * It is a subset of allApns and has the same format
+ */
+private ArrayList<ApnSetting> waitingApns = null;
+
+private ApnSetting preferredApn = null;
+
+/* Currently active APN */
+protected ApnSetting mActiveApn;
+static          final          Uri          PREFERAPN_URI          =
Uri.parse("content://telephony/carriers/preferapn");
+static final String APN_ID = "apn_id";
+private boolean canSetPreferApn = false;
+protected static final int APN_DELAY_MILLIS = 5000;

 /**
   * dataConnectionList holds all the Data connection
/** Currently active CdmaDataConnection */
private CdmaDataConnection mActiveDataConnection;
+/** mimic of GSM's mActiveApn */
+private boolean mIsApnActive = false;
private boolean mPendingRestartRadio = false;
private static final int TIME_DELAYED_TO_RESTART_RADIO =
          SystemProperties.getInt("ro.cdma.timetoradiorestart", 60000);
```

In the function named CdmaDataConnectionTracker(CDMAPhone p):

```
  mDataConnectionTracker = this;

createAllDataConnectionList();
+ apnObserver = new ApnChangeObserver();
+p.getContext().getContentResolver().registerContentObserver(
+Telephony.Carriers.CONTENT_URI, true, apnObserver);
```

```
        // This preference tells us 1) initial condition for "dataEnabled",
        // and 2) whether the RIL will setup the baseband to auto-PS attach.
```

In function: public void dispose()

```
        mCdmaPhone.mSST.unregisterForRoamingOn(this);
        mCdmaPhone.mSST.unregisterForRoamingOff(this);
        phone.mCM.unregisterForCdmaOtaProvision(this);
       + phone.getContext().getContentResolver().unregisterContentObserver(this.apnObserver);
        phone.getContext().unregisterReceiver(this.mIntentReceiver);
        destroyAllDataConnectionList();

     @Override

+protected boolean isApnTypeAvailable(String type) {
+       if (allApns != null) {
+           for (ApnSetting apn : allApns) {
+               if (apn.canHandleType(type)) {
+                       return true;
+               }
+           }
+       }
+       return false;
   }
   protected String[] getActiveApnTypes() {
```

In the function named getActiveApnTypes:

```
     protected String[] getActiveApnTypes() {
    String[] result;
    - if (mActiveApn != null) {
    - result = mActiveApn.types;
    + if (mIsApnActive) {
    +    result = mSupportedApnTypes.clone();
      } else {
        result = new String[1];
        result[0] = Phone.APN_TYPE_DEFAULT;
```

In the function named protected String getActiveApnString():

```
    +protected String getActiveApnString() {
    +    String result = null;
    +    if (mActiveApn != null) {
    +            result = mActiveApn.apn;
    +    }
    +    return result;
    +}
```

```
+
/**
* The data connection is expected to be setup while device
```

In the function named private boolean trySetupData(String reason):

```
          && (psState == ServiceState.STATE_IN_SERVICE)
          && ((phone.mCM.getRadioState() == CommandsInterface.RadioState.NV_READY) ||
                   mCdmaPhone.mRuimRecords.getRecordsLoaded())
-         && (mCdmaPhone.mSST.isConcurrentVoiceAndData() ||
-                  phone.getState() == Phone.State.IDLE )
          && isDataAllowed()
          && desiredPowerState
          && !mPendingRestartRadio
          && !mCdmaPhone.needsOtaServiceProvisioning()) {

+if (state == State.IDLE) {
+      waitingApns = buildWaitingApns();
+      if (waitingApns.isEmpty()) {
+            if (DBG) log("No APN found");
+            notifyNoData(CdmaDataConnection.FailCause.MISSING_UKNOWN_APN);
+            return false;
+      } else {
+            log ("Create from allApns : " + apnListToString(allApns));
+      }
+}

+if (DBG) {
+      log ("Setup waitngApns : " + apnListToString(waitingApns));
+}

    return setupData(reason);

    } else {
```

In the function named private void cleanUpConnection:

```
    stopNetStatPoll();

-if (!notificationDeferred) {
-     if (DBG) log("cleanupConnection: !notificationDeferred");
-          gotoIdleAndNotifyDataConnection(reason);
+           if (!tearDown) {
+                setState(State.IDLE);
+                phone.notifyDataConnection(reason);
+                mActiveApn = null;     // ZTE_CDMAAPN_ZJ_001,   2010-06-10
+               mIsApnActive = false;
            }
        }
```

In the function named private boolean setupData(String reason):

```
    private boolean setupData(String reason) {
+ApnSetting apn= getNextApn();
+if (apn == null) return false;
        CdmaDataConnection conn = findFreeDataConnection();
if (conn == null) {
  if (DBG) log("setupData: No free CdmaDataConnectionfound!");
                    return false;
                }
+mActiveApn = apn;
+mIsApnActive = true;
  mActiveDataConnection = conn;
  String[] types;
  if (mRequestedApnType.equals(Phone.APN_TYPE_DUN)) {
        types = new String[1];
        types[0] = Phone.APN_TYPE_DUN;
} else {
        types = mDefaultApnTypes;
}
-mActiveApn = new ApnSetting(0, "", "", "", "", "", "", "", "", "", "", 0, types);
Message msg = obtainMessage();
  msg.what = EVENT_DATA_SETUP_COMPLETE;
  msg.obj = reason;
-conn.connect(msg, mActiveApn);
+conn.connect(msg, apn);

  setState(State.INITING);
  phone.notifyDataConnection(reason);
```

In the function named onRecordsLoaded:

```
        protected void onRecordsLoaded() {
+               createAllApnList();
                if (state == State.FAILED) {
                        cleanUpConnection(false, null);
```

In the function named onEnableNewApn:

```
protected void onEnableNewApn() {
+mRetryMgr.resetRetryCount();
+ Log.d(LOG_TAG, "onEnableNewApn " );
```

```
      cleanUpConnection(true, Phone.REASON_APN_SWITCHED);
```

In the function named onDataSetupComplete:

```
    if (ar.exception == null) {
    +
    +
    +         // everything is setup
    +         if (isApnTypeActive(Phone.APN_TYPE_DEFAULT)) {
    +              SystemProperties.set("cdma.defaultDatacontext.active", "true");
    +                        if (canSetPreferApn && preferredApn == null) {
    +                             Log.d(LOG_TAG, "PREFERED APN is null");
    +                             preferredApn = mActiveApn;
    +                             setPreferredApn(preferredApn.id);
    +                        }
    +         }  dse  {
    +              SystemProperties.set("gsm.defaultpdpcontext.active", "false");
    +         }
    +         notifyDefaultData(reason);
    +
    +         // TODO: For simultaneous PDP support, we need to build another
    +         // trigger another TRY_SETUP_DATA for the next APN type.    (Note
    +         // that the existing connection may service that type, in which
    +         // case we should try the next type, etc.
    + } else {
    +         CdmaDataConnection.FailCause cause;
    +         cause = (CdmaDataConnection.FailCause) (ar.result);
    +         if(DBG) log("cdma data setup failed " + cause);
    +                   // Log this failure to the Event Logs.
    +         if (cause.isEventLoggable()) {
    +              int cid = -1;
    +              //CdmaCellLocation loc = ((CdmaCellLocation)phone.getCellLocation());
    +             // if (loc != null) cid = loc.getCid();
    +
    +            //    EventLog.List val = new EventLog.List(
    +             //              cause.ordinal(), cid,
    +                //           TelephonyManager.getDefault().getNetworkType());
    +                                                                                              //
    EventLog.writeEvent(TelephonyEventLog.EVENT_LOG_RADIO_PDP_SETUP_FAIL, val);
    +         }
    +
    +         // No try for permanent failure
    +         if (cause.isPermanentFail()) {
    +              notifyNoData(cause);
    +              if (!mRequestedApnType.equals(Phone.APN_TYPE_DEFAULT)) {
```

```
+                   phone.notifyDataConnection(Phone.REASON_APN_FAILED);
+                   onEnableApn(apnTypeToId(mRequestedApnType), DISABLED);
+               }
+               return;
+           }
+
+           waitingApns.remove(0);
+           if (waitingApns.isEmpty()) {
+               // No more to try, start delayed retry
+               startDelayedRetry(cause, reason);
+           } else {
+               // we still have more apns to try
+               setState(State.SCANNING);
+               // Wait a bit before trying the next APN, so that
+               // we're not tying up the RIL command channel
+                   sendMessageDelayed(obtainMessage(EVENT_TRY_SETUP_DATA, reason),
APN_DELAY_MILLIS);
+           }
+ }
}
/**
 * Called when EVENT_DISCONNECT_DONE is received.
```

In the function named protected void onDisconnectDone(AsyncResult ar), add the following code:

```
                phone.notifyDataConnection(reason);
+               mIsApnActive = false;
                mActiveApn = null;
                if (retryAfterDisconnected(reason)) {
```

In the function named public void handleMessage (Message msg), add the following code:

```
        onRestartRadio();
        break;

+ case EVENT_APN_CHANGED:
+     onApnChanged();
+     break;

      default:
```

The follows are all new added functions.

```
+/**
+ * Based on the sim operator numeric, create a list for all possible pdps
+ * with all apns associated with that pdp
+ *
+ *
+ */
+private void createAllApnList() {
+       allApns = new ArrayList<ApnSetting>();
+       String operator = mCdmaPhone.mRuimRecords.getRUIMOperatorNumeric();
+
+       if (operator != null) {
+           String selection = "numeric = '" + operator + "'";
+
+           Cursor cursor = phone.getContext().getContentResolver().query(
+                   Telephony.Carriers.CONTENT_URI, null, selection, null, null);
+
+           if (cursor != null) {
+               if (cursor.getCount() > 0) {
+                   allApns = createApnList(cursor);
+                   // TODO: Figure out where this fits in.   This basically just
+                   // writes the pap-secrets file.   No longer tied to PdpConnection
+                   // object.   Not used on current platform (no ppp).
+                   //PdpConnection pdp = pdpList.get(pdp_name);
+                   //if (pdp != null && pdp.dataLink != null) {
+                   //       pdp.dataLink.setPasswordInfo(cursor);
+                   //}
+               }
+               cursor.close();
+           }
+       }
+
+       if (allApns.isEmpty()) {
+           if (DBG) log("No APN found for carrier: " + operator);
+           preferredApn = null;
+           notifyNoData(CdmaDataConnection.FailCause.MISSING_UKNOWN_APN);
+       } else {
+           preferredApn = getPreferredApn();
+           Log.d(LOG_TAG, "Get PreferredAPN");
+           if (preferredApn != null && !preferredApn.numeric.equals(operator)) {
+               preferredApn = null;
+               setPreferredApn(-1);
+           }
+       }
+}
+
+
+private ArrayList<ApnSetting> createApnList(Cursor cursor) {
+       ArrayList<ApnSetting> result = new ArrayList<ApnSetting>();
+       if (cursor.moveToFirst()) {
+               do {
+                       String[] types = parseTypes(
+
cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.TYPE)));
+                       ApnSetting apn = new ApnSetting(
+ cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers._ID)),
+cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.NUMERIC)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.NAME)),
+cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.APN)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PROXY)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PORT)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSC)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSPROXY)),
+cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSPORT)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.USER)),
+ cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PASSWORD)),
+ cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers.AUTH_TYPE)),
+                               types);
+                       result.add(apn);
+               } while (cursor.moveToNext());
+       }
+       return result;
+}


+private void setPreferredApn(int pos) {
+    if (!canSetPreferApn) {
+        return;
+    }
+
+    ContentResolver resolver = phone.getContext().getContentResolver();
+    resolver.delete(PREFERAPN_URI, null, null);
+
+    if (pos >= 0) {
+        ContentValues values = new ContentValues();
+        values.put(APN_ID, pos);
+        resolver.insert(PREFERAPN_URI, values);
+    }
+}
+
+private String[] parseTypes(String types) {
+       String[] result;
+       // If unset, set to DEFAULT.
+       if (types == null || types.equals("")) {
+               result = new String[1];
+               result[0] = Phone.APN_TYPE_ALL;
+       } else {
+               result = types.split(",");
+       }
+       return result;
+}
+


+private String apnListToString (ArrayList<ApnSetting> apns) {
+       StringBuilder result = new StringBuilder();
+       for (int i = 0, size = apns.size(); i < size; i++) {
+               result.append('[')
+                             .append(apns.get(i).toString())
+                             .append(']');
+       }
+       return result.toString();
+}
+
+


+private ApnSetting getNextApn() {
+       ArrayList<ApnSetting> list = waitingApns;
+       ApnSetting apn = null;
+
+       if (list != null) {
+               if (!list.isEmpty()) {
+                       apn = list.get(0);
+               }
+       }
+       return apn;
+}


+private ApnSetting getPreferredApn() {
+       if (allApns.isEmpty()) {
```

```
+             return null;
+       }
+
+       Cursor cursor = phone.getContext().getContentResolver().query(
+                   PREFERAPN_URI, new String[] { "_id", "name", "apn" },
+                   null, null, Telephony.Carriers.DEFAULT_SORT_ORDER);
+
+       if (cursor != null) {
+             canSetPreferApn = true;
+       }  else  {
+             canSetPreferApn = false;
+       }
+
+       if (canSetPreferApn && cursor.getCount() > 0) {
+             int pos;
+             cursor.moveToFirst();
+             pos = cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers._ID));
+             for(ApnSetting p:allApns) {
+                   if (p.id == pos && p.canHandleType(mRequestedApnType)) {
+                         cursor.close();
+                         return p;
+                   }
+             }
+       }
+
+       if (cursor != null) {
+             cursor.close();
+       }
+
+       return null;
+}
+/**
+ *
+ * @return waitingApns list to be used to create PDP
+ *                error when waitingApns.isEmpty()
+ */


+private ArrayList<ApnSetting> buildWaitingApns() {
+       ArrayList<ApnSetting> apnList = new ArrayList<ApnSetting>();
+       String operator = mCdmaPhone.mRuimRecords.getRUIMOperatorNumeric();
+       Log.i(LOG_TAG, "AbuildWaitingApns operator=" + operator);
+       if (mRequestedApnType.equals(Phone.APN_TYPE_DEFAULT)) {
+ Log.i(LOG_TAG, "APN    get APN_TYPE_DEFAULT");
+           if (canSetPreferApn && preferredApn != null) {
+               Log.i(LOG_TAG, "Preferred APN:" + operator + ":"
+                       + preferredApn.numeric + ":" + preferredApn);
+               if (preferredApn.numeric.equals(operator)) {
+                   Log.i(LOG_TAG, "Waiting APN set to preferred APN");
+                   apnList.add(preferredApn);
+                   return apnList;
+               } dse {
+                   setPreferredApn(-1);
+                   preferredApn = null;
+               }
+           }
+       }
+
+       if (allApns != null) {
+           for (ApnSetting apn : allApns) {
+               if (apn.canHandleType(mRequestedApnType)) {
+                   apnList.add(apn);
+               }
+           }
+       }
+       return apnList;
+}
+/**
+ * Checks if the PreferredApn is Changed
+ **/


+private boolean IsPreferredApnChanged() {
+      boolean change = true;
+      Log.d(LOG_TAG, "mActiveApn: "+ mActiveApn);
+      Log.d(LOG_TAG, "Preferred APN: " +    preferredApn );
+
+      if ((preferredApn != null) && (mActiveApn != null))    {
+          if ((mActiveApn.toString().equals(preferredApn.toString() )) &&
+              ((mActiveApn.user != null && mActiveApn.user.equals(preferredApn.user)) ||
+               (mActiveApn.user == null && preferredApn.user == null)) &&
+                                    ((mActiveApn.password    !=    null    &&
mActiveApn.password.equals(preferredApn.password)) ||
+               (mActiveApn.password == null && preferredApn.password == null)))    {
+               change = false;
+          }
+      }
+      Log.d(LOG_TAG, "Is Preferred APN changed: "+ change);
+      return change;
+}


+ private void onApnChanged() {
+        boolean isConnected;
+
+        isConnected = (state != State.IDLE && state != State.FAILED);
+
+        // The "current" may no longer be valid.    MMS depends on this to send properly.
+      String operator = mCdmaPhone.mRuimRecords.getRUIMOperatorNumeric();
+
+        mCdmaPhone.updateCurrentCarrierInProvider(operator);
+
+        // TODO: It'd be nice to only do this if the changed entrie(s)
+        // match the current operator.
+        createAllApnList();
+        if (IsPreferredApnChanged()) {
+
+
+              if (state != State.DISCONNECTING) {
+                    cleanUpConnection(isConnected, Phone.REASON_APN_CHANGED);
+                    if (!isConnected) {
+                          //  reset reconnect timer
+                          mRetryMgr.resetRetryCount();
+                          //mReregisterOnReconnectFailure = false;
+                          trySetupData(Phone.REASON_APN_CHANGED);
+                  }
+            }
+      }
+}
+
+


+// get ApnSetting by apnId
+private ApnSetting getProjectionByApnId(String apnId) {
+     ApnSetting apnSetting = null;
+
+     String where = "_id = " + apnId;
+
+     Cursor cursor = phone.getContext().getContentResolver().query(
+                              Telephony.Carriers.CONTENT_URI,
+                              new String[] { Telephony.Carriers._ID,
+                                Telephony.Carriers.NUMERIC,
+                                Telephony.Carriers.NAME,
+                                Telephony.Carriers.APN,
+                                Telephony.Carriers.PROXY,
+                                Telephony.Carriers.PORT,
+                                Telephony.Carriers.MMSC,
+                                Telephony.Carriers.MMSPROXY,
+                                Telephony.Carriers.MMSPORT,
+                                Telephony.Carriers.USER,
+                                Telephony.Carriers.PASSWORD,
+                                Telephony.Carriers.AUTH_TYPE,
+                                Telephony.Carriers.TYPE }, where, null,
+                       Telephony.Carriers.DEFAULT_SORT_ORDER);
+
+    if (cursor.getCount() > 0) {
+         cursor.moveToFirst();
+
+         String[]                              types                              =
parseTypes(cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.TYPE)));
+
+         apnSetting = new ApnSetting(
+     cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers._ID)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.NUMERIC)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.NAME)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.APN)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PROXY)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PORT)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSC)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSPROXY)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSPORT)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.USER)),
+    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PASSWORD)),
+    cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers.AUTH_TYPE)),
+                   types);
+    }
+
+    return apnSetting;
+}


+// set pppd Apn File (path : /data/local/pppd_apn)
+private void setPppdApnFile(String key) {
+    Log.d(LOG_TAG, "wsh : key : " + key);
+    ApnSetting apnSetting = getProjectionByApnId(key);
+    Log.d(LOG_TAG, "wsh : user : " + apnSetting.user);
+
+    if ((apnSetting.user.equals("ctwap@mycdma.cn"))
+                || (apnSetting.user.equals("ctnet@mycdma.cn"))) {
+
+          try {
+                Log.d(LOG_TAG, "wsh : system call /system/etc/ppp/set_pppd_apn.sh");
+
+                // ZTE_WSH_C+W_100710,begin
+                String proxyStr = "";
+
+                Log.d(LOG_TAG,  "wsh  :  apnSetting.proxy  =  "  +  apnSetting.proxy  +  "
apnSetting.port = " + apnSetting.port);
+
+                if( (apnSetting.proxy == null) || (apnSetting.port == null)) {
+                      Log.d(LOG_TAG, "wsh : proxy = null don't set proxy");
+                      proxyStr = "";
+                } else {
+                      if((apnSetting.proxy.equals("")) || (apnSetting.port.equals(""))) {
+                            Log.d(LOG_TAG, "wsh : don't set proxy");
+                            proxyStr = "";
+                      } else {
+                            Log.d(LOG_TAG, "wsh : set proxy");
+                            proxyStr = "http://" + apnSetting.proxy + ":" + apnSetting.port +
"/";
+                      }
+                }
+
+                Log.d(LOG_TAG, "wsh : proxyStr = " + proxyStr);
+                Runtime.getRuntime().exec("/system/etc/ppp/set_pppd_apn.sh          "          +
apnSetting.user + " " + proxyStr);
+                // ZTE_WSH_C+W_100710,end
+          } catch (IOException e) {
+                // TODO Auto-generated catch block
+                e.printStackTrace();
+          }
+    }
+}
+
  }
```

In this file, the functions of building APN list and the processing of APN when changed are added. The emphasis of the modification is in the fuction named setupData. Every modifications are around this emphasis.

#### 6.2.2.3 ApnSetting.java File Modification

In the file frameworks\base\telephony\java\com\android\internal\telephony\gsm\ApnSetting.java, because the CDMA branch needs to use the content of apnsetting.java, so change all common variables to pullic variables.

For examples:

```
String carrier;      ->      public   String carrier;
String apn;          ->      public   String apn;
String proxy;        ->      public   String proxy;
String port;         ->      public   String port;
String mmsc;         ->      public   String mmsc;
String mmsProxy;     ->      public   String mmsProxy;
String mmsPort;      ->      public   String mmsPort;
String user;         ->      public   String user;
String password;     ->      public   String password;
```

## 6.3 Android4.0 CDMA branch increases APN settings

### 6.3.1 The modification of CdmaDataConnection.java

In the class named CdmaDataConnection at the file of CdmaDataConnection.java, you should add the following content.

The modification is:

In the function of "phone.mCM.setupDataCall" locate in the class "onConnect", change the three parameters of "NULL" to "cp.apn.apn, cp.apn.user,cp.apn.password,"

```
phone.mCM.setupDataCall(Integer.toString(RILConstants.SETUP_DATA_TECH_CDMA),
                Integer.toString(RILConstants.DATA_PROFILE_DEFAULT), NULL, NULL
                NULL, Integer.toString(RILConstants.SETUP_DATA_AUTH_PAP_CHAP),
                RILConstants.SETUP_DATA_PROTOCOL_IP,msg);

phone.mCM.setupDataCall(Integer.toString(RILConstants.SETUP_DATA_TECH_CDMA),
                Integer.toString(RILConstants.DATA_PROFILE_DEFAULT), cp.apn.apn,
cp.apn.user,
cp.apn.password, Integer.toString(RILConstants.SETUP_DATA_AUTH_PAP_CHAP),
                RILConstants.SETUP_DATA_PROTOCOL_IP,msg);
```

### 6.3.2 The modification of CdmaDataConnectionTracker.java

Add code as follows:

```
boolean failNextConnect = false;
+ /*ZTE _CDMAAPN_001 begin*/
+      private class ApnChangeObserver extends ContentObserver {
```

```
+               public ApnChangeObserver () {
+                       super(mDataConnectionTracker);
+               }
+               @Override
+               public void onChange(boolean selfChange) {
+                       sendMessage(obtainMessage(EVENT_APN_CHANGED));
+               }
+       }
+       private ArrayList<ApnSetting> allApns = null;
+       private ApnChangeObserver apnObserver;
+       private ArrayList<ApnSetting> waitingApns = null;
+       private ApnSetting preferredApn = null;

+       protected ApnSetting mActiveApn;
+       static final Uri PREFERAPN_URI = Uri.parse("content://telephony/carriers/preferapn");
+       static final String APN_ID = "apn_id";
+       private boolean canSetPreferApn = false;
+       protected static final int APN_DELAY_MILLIS = 5000;
+ /*ZTE _CDMAAPN_001 end*/
```

In the function named CdmaDataConnectionTracker, add code as follows:

```
                createAllDataConnectionList();
+ /*ZTE _CDMAAPN_001 begin*/
+               apnObserver = new ApnChangeObserver();
+               p.getContext().getContentResolver().registerContentObserver(
+                               Telephony.Carriers.CONTENT_URI, true, apnObserver);
+ /*ZTE _CDMAAPN_001 end*/
```

In the function named dispose, add the code as follows:

```
+ mPhone.getContext().getContentResolver().unregisterContentObserver(this.apnObserver);
+ /*ZTE _CDMAAPN_001*/
                destroyAllDataConnectionList();
```

In the function named setState, add the code as follows:

```
+               //mState = s;
+       }
+               mState = s;
+       }
```

After the function named setState, add the code as follows:

```
+ /*ZTE _ CDMAAPN _001 begin*/
+       @Override
+       protected boolean isApnTypeActive(String type) {
+               return mActiveApn != null && mActiveApn.canHandleType(type);
+       }
+ /*ZTE _ CDMAAPN _001 end*/
```

In the function named isApnTypeAvailable, add the code as follows:

```
+     protected boolean isApnTypeAvailable(String type) {
+             if (allApns != null) {
+                     for (ApnSetting apn : allApns) {
+                             if (apn.canHandleType(type)) {
+                                     return true;
+                             }
+                     }
+             }
+             return false;
+     }
```

In the function named trySetupData, add the code as follows:

```
          if ((mState == State.IDLE || mState == State.SCANNING) &&
                  isDataAllowed() && getAnyDataEnabled() && !isEmergency()) {
+/*ZTE _CDMAAPN_001 begin*/
+       log("CDMA APN     trySetupData (mState == State.IDLE || mState == State.SCANNING)
+&& in");
+                     if (mState == State.IDLE) {
+                             waitingApns = buildWaitingApns();
+                             if (waitingApns.isEmpty()) {
+                                     if (DBG) log("No APN found");
+ log("CDMA APN onDataSetupComplete    No APN found");
+ notifyNoData(CdmaDataConnection.FailCause.MISSING_UNKNOWN_APN);
+                                     return false;
+                             } else {
+                               log("CDMA APN onDataSetupComplete     APN found");
+                             //      log ("Create from allApns : " + apnListToString(allApns));
+                             }
+                     }
+                     boolean retValue = setupData(reason);
+                     notifyOffApnsOfAvailability(reason);
+                     if (DBG) {
+                             log ("Setup waitngApns : " );
+                     }
+/*ZTE_JJH_CDMAAPN_001 end*/
                     return retValue;
```

In the function named cleanUpConnection, add the code as follows:

```
           mPhone.notifyDataConnection(reason);
+          mActiveApn = null;     // ZTE_CDMAAPN_JJH_001
           mIsApnActive = false;
```

In the function named setupData, add the code as follows:

```
+/*ZTE _CDMAAPN_001 begin*/
+ log("CDMA APN setupData");
+           ApnSetting apn= getNextApn();
+       if (apn == null)    return false;
+       log("CDMA APN setupData apn != null");
+ /*ZTE _CDMAAPN_001 end*/
            CdmaDataConnection conn = findFreeDataConnection();

            if (conn == null) {
                log("CDMA APN setupData conn == null");
                if (DBG) log("setupData: No free CdmaDataConnection found!");
                return false;
            }

          /** TODO: We probably want the connection being setup to a parameter passed around */
+             mActiveApn = apn;/*ZTE_JJH_CDMAAPN_001*/
+             mIsApnActive = true;/*ZTE_JJH_CDMAAPN_001*/
```

Modify the code as follow:

```
+ conn.bringUp(msg, apn);
//conn.connect(msg, mActiveApn);
setState(State.INITING);
```

In the function named onRecordsLoaded, add the code as follows:

```
createAllApnList();/*ZTE_CDMAAPN_001*/
if (mState == State.FAILED) {
```

In the function named onDataSetupComplete, modify the code as follows:

```
protected void onDataSetupComplete(AsyncResult ar)
{
            String reason = null;
            if (ar.userObj instanceof String) {
                reason = (String) ar.userObj;
            }
 log("CDMA APN onDataSetupComplete"+reason);
            if (ar.exception == null) {

 log("CDMA APN onDataSetupComplete    ar.exception == null ");
                // everything is setup
                if (isApnTypeActive(Phone.APN_TYPE_DEFAULT)) {
                    SystemProperties.set("cdma.defaultDatacontext.active", "true");
 log("CDMA APN onDataSetupComplete    cdma.defaultDatacontext.active ");
```

```
                        if (canSetPreferApn && preferredApn == null) {
                            Log.d(LOG_TAG, "PREFERED APN is null");
                            preferredApn = mActiveApn;
 log("CDMA APN onDataSetupComplete    setPreferredApn(preferredApn.id) ");
                            setPreferredApn(preferredApn.id);
                        }
                } else {
                  log("CDMA APN onDataSetupComplete cdma.defaultpdpcontext.active false");
                    SystemProperties.set("cdma.defaultpdpcontext.active", "false");
                }
                notifyDefaultData(reason);

                // TODO: For simultaneous PDP support, we need to build another
                // trigger another TRY_SETUP_DATA for the next APN type.    (Note
                // that the existing connection may service that type, in which
                // case we should try the next type, etc.
            } else {
                CdmaDataConnection.FailCause cause;
                  log("CDMA APN onDataSetupComplete    ar.exception != null ");
                cause = (CdmaDataConnection.FailCause) (ar.result);
                if(DBG) log("cdma data setup failed " + cause);
                        // Log this failure to the Event Logs.
                if (cause.isEventLoggable()) {
                    int cid = -1;
                    //CdmaCellLocation loc = ((CdmaCellLocation)phone.getCellLocation());
                  // if (loc != null) cid = loc.getCid();
                  log("CDMA APN onDataSetupComplete    cause.isEventLoggable()");
                   //    EventLog.List val = new EventLog.List(
                  //              cause.ordinal(), cid,
                    //          TelephonyManager.getDefault().getNetworkType());
                  //
EventLog.writeEvent(TelephonyEventLog.EVENT_LOG_RADIO_PDP_SETUP_FAIL, val);
                }

                // No try for permanent failure
                if (cause.isPermanentFail()) {
                    notifyNoData(cause);
 log("CDMA APN onDataSetupComplete    cause.isPermanentFail()");
                    if (!mRequestedApnType.equals(Phone.APN_TYPE_DEFAULT)) {
                        mPhone.notifyDataConnection(Phone.REASON_APN_FAILED);
 log("CDMA APN onDataSetupComplete
mRequestedApnType.equals(Phone.APN_TYPE_DEFAULT)");
                        onEnableApn(apnTypeToId(mRequestedApnType), DISABLED);
                    }
```

```
                        return;
                }

                waitingApns.remove(0);
                if (waitingApns.isEmpty()) {
 log("CDMA APN onDataSetupComplete    waitingApns.isEmpty");
                        // No more to try, start delayed retry
                        startDelayedRetry(cause, reason);
                } else {
                        // we still have more apns to try
                        setState(State.SCANNING);
 log("CDMA APN onDataSetupComplete    State.SCANNING");
                        // Wait a bit before trying the next APN, so that
                        // we're not tying up the RIL command channel
                        sendMessageDelayed(obtainMessage(EVENT_TRY_SETUP_DATA, reason),
APN_DELAY_MILLIS);
                }
        }
    }
```

In the function named handleMessage, add the code as follows:

```
                        break;
+    /*ZTE _CDMAAPN_001 begin*/
+                   case EVENT_APN_CHANGED:
+                           log("CDMA APN EVENT_APN_CHANGED    ");
+                           onApnChanged();
+                           break;
+    /*ZTE _CDMAAPN_001 end*/
                default:
```

At the end of this file, add the code as follows:

```
/*ZTE _CDMAAPN_001 begin*/

    /**
     * Based on the sim operator numeric, create a list for all possible pdps
     * with all apns associated with that pdp
     *
     *
     */
private void createAllApnList() {
        allApns = new ArrayList<ApnSetting>();
        String operator = mCdmaPhone.mIccRecords.getOperatorNumeric();
        log("CDMA APN      createAllApnList");
        if (operator != null) {
                String selection = "numeric = '" + operator + "'";
                log("CDMA APN      createAllApnList     operator != null");
                Cursor cursor = mPhone.getContext().getContentResolver().query(
                        Telephony.Carriers.CONTENT_URI, null, selection, null, null);

                if (cursor != null) {
                    log("CDMA APN      createAllApnList     cursor != null");
                    if (cursor.getCount() > 0) {
                        log("CDMA APN      createAllApnList     cursor.getCount() > 0");
                        allApns = createApnList(cursor);
                        // TODO: Figure out where this fits in.    This basically just
                        // writes the pap-secrets file.    No longer tied to PdpConnection
                        // object.    Not used on current platform (no ppp).
                        //PdpConnection pdp = pdpList.get(pdp_name);
                        //if (pdp != null && pdp.dataLink != null) {
                        //        pdp.dataLink.setPasswordInfo(cursor);
                        //}
                    }
                    cursor.close();
                }
            }

            if (allApns.isEmpty()) {
                log("CDMA APN      createAllApnList     allApns.isEmpty()");
                if (DBG) log("No APN found for carrier: " + operator);
                preferredApn = null;
                notifyNoData(CdmaDataConnection.FailCause.MISSING_UNKNOWN_APN);
            }  else  {
                log("CDMA APN      createAllApnList     allApns.is not Empty");
                preferredApn = getPreferredApn();
                Log.d(LOG_TAG, "Get PreferredAPN");
                if (preferredApn != null && !preferredApn.numeric.equals(operator)) {
                    preferredApn = null;
                log("CDMA APN      createAllApnList     setPreferredApn(-1)");
                    setPreferredApn(-1);
                }
            }
        }


    private ArrayList<ApnSetting> createApnList(Cursor cursor) {
            ArrayList<ApnSetting> result = new ArrayList<ApnSetting>();
            log("CDMA APN      createApnList");
            if (cursor.moveToFirst()) {
                log("CDMA APN     createApnList cursor.moveToFirst()");
                do {
                    String[] types = parseTypes(

    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.TYPE)));
                    ApnSetting apn = new ApnSetting(

    cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers._ID)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.NUMERIC)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.NAME)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.APN)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PROXY)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PORT)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSC)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSPROXY)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.MMSPORT)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.USER)),
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PASSWORD)),
    cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers.AUTH_TYPE)),
                                            types,
    cursor.getString(cursor.getColumnIndexOrThrow(Telephony.Carriers.PROTOCOL)),
                                            cursor.getString(cursor.getColumnIndexOrThrow(
                                                    Telephony.Carriers.ROAMING_PROTOCOL)),
                                            cursor.getInt(cursor.getColumnIndexOrThrow(
                                                    Telephony.Carriers.CARRIER_ENABLED)) == 1,
    cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers.BEARER)));
                    result.add(apn);
                } while (cursor.moveToNext());
            }
            return result;
        }


    private void setPreferredApn(int pos) {
            if (!canSetPreferApn) {
                log("CDMA APN can not setPreferredApn");
                return;
            }
            log("CDMA APN setPreferredApn");
            ContentResolver resolver = mPhone.getContext().getContentResolver();
            resolver.delete(PREFERAPN_URI, null, null);

            if (pos >= 0) {
                log("CDMA APN setPreferredApn pos ="+pos);
                ContentValues values = new ContentValues();
                values.put(APN_ID, pos);
                resolver.insert(PREFERAPN_URI, values);
            }
        log("CDMA APN setPreferredApn pos ="+pos);
      }

private String[] parseTypes(String types) {
            String[] result;
            log("CDMA APN parseTypes");
            // If unset, set to DEFAULT.
            if (types == null || types.equals("")) {
                log("CDMA APN parseTypes    if ");
                result = new String[1];
                result[0] = Phone.APN_TYPE_ALL;
            } else {
                log("CDMA APN parseTypes else");
                result = types.split(",");
            }
             log("CDMA APN parseTypes result "+result);
            return result;
      }

private String apnListToString (ArrayList<ApnSetting> apns) {
            StringBuilder result = new StringBuilder();
             log("CDMA APN apnListToString");
            for (int i = 0, size = apns.size(); i < size; i++) {
                result.append('[')
                      .append(apns.get(i).toString())
                      .append(']');
            }
             log("CDMA APN apnListToString result"+result.toString());
            return result.toString();
      }


private ApnSetting getNextApn() {
            ArrayList<ApnSetting> list = waitingApns;
            ApnSetting apn = null;
             log("CDMA APN getNextApn");
            if (list != null) {
                 log("CDMA APN getNextApn list != null");
                if (!list.isEmpty()) {
                    log("CDMA APN getNextApn list is not empty");
                    apn = list.get(0);
                }
            }
            log("CDMA APN getNextApn list is "+apn);
            return apn;
        }

private ApnSetting getPreferredApn() {
            if (allApns.isEmpty()) {
                log("CDMA APN getPreferredApn is empty");
                return null;
            }
             log("CDMA APN getPreferredApn");
            Cursor cursor = mPhone.getContext().getContentResolver().query(
                    PREFERAPN_URI, new String[] { "_id", "name", "apn" },
                    null, null, Telephony.Carriers.DEFAULT_SORT_ORDER);

            if (cursor != null) {
                canSetPreferApn = true;
            } else {
                canSetPreferApn = false;
            }
             log("CDMA APN getPreferredApn canSetPreferApn ="+canSetPreferApn );
            if (canSetPreferApn && cursor.getCount() > 0) {
                int pos;
                cursor.moveToFirst();
                log("CDMA APN getNextApn cursor.moveToFirst()");
                pos = cursor.getInt(cursor.getColumnIndexOrThrow(Telephony.Carriers._ID));
                for(ApnSetting p:allApns) {
                    if (p.id == pos && p.canHandleType(mRequestedApnType)) {
                        cursor.close();
                        return p;
                    }
                }
            }

            if (cursor != null) {
                  log("CDMA APN getPreferredApn cursor != null");
                cursor.close();
            }

            return null;
        }
        /**
        *
        * @return waitingApns list to be used to create PDP
        *                error when waitingApns.isEmpty()
        */
private ArrayList<ApnSetting> buildWaitingApns() {
            ArrayList<ApnSetting> apnList = new ArrayList<ApnSetting>();
            String operator = mCdmaPhone.mIccRecords.getOperatorNumeric();
            Log.i(LOG_TAG, "AbuildWaitingApns operator=" + operator);
            if (mRequestedApnType.equals(Phone.APN_TYPE_DEFAULT)) {
                  Log.i(LOG_TAG, "APN    get APN_TYPE_DEFAULT");
                  log("CDMA APN buildWaitingApnsl");
                  if (canSetPreferApn && preferredApn != null) {
                               Log.i(LOG_TAG, "Preferred APN:" + operator + ":"
                               + preferredApn.numeric + ":" + preferredApn);
                        if (preferredApn.numeric.equals(operator)) {

                              apnList.add(preferredApn);
                              Log.i(LOG_TAG, "Waiting APN set to preferred APN"+apnList);

                              return apnList;
                        } else {
                              setPreferredApn(-1);
                              preferredApn = null;
                        }
                  }
            }

            if (allApns != null) {
                  log("CDMA APN buildWaitingApns allApns != null");
                  for (ApnSetting apn : allApns) {
                        if (apn.canHandleType(mRequestedApnType)) {
                              apnList.add(apn);
                        }
                  }
            }
            log("CDMA APN buildWaitingApns allApns apnList");
            return apnList;
      }
      /**
        * Checks if the PreferredApn is Changed
        **/
private boolean IsPreferredApnChanged() {
           boolean change = true;
           Log.d(LOG_TAG, "mActiveApn: "+ mActiveApn);
           Log.d(LOG_TAG, "Preferred APN: " +   preferredApn );
           if ((preferredApn != null) && (mActiveApn != null))   {
               log("CDMA APN IsPreferredApnChanged preferredApn != null");
               if ((mActiveApn.toString().equals(preferredApn.toString() )) &&
                   ((mActiveApn.user != null && mActiveApn.user.equals(preferredApn.user)) ||
                        (mActiveApn.user == null && preferredApn.user == null)) &&
                        ((mActiveApn.password != null &&
                        mActiveApn.password.equals(preferredApn.password)) ||
                        (mActiveApn.password == null && preferredApn.password == null)))   {
                        change = false;
                        log("CDMA APN IsPreferredApnChanged     change = false;");
                  }
           }
           Log.d(LOG_TAG, "Is Preferred APN changed: "+ change);
           return change;
      }

  private void onApnChanged() {
            boolean isConnected;
            Log.i(LOG_TAG, " test onApnChanged");
            isConnected = (mState != State.IDLE && mState != State.FAILED);

            // The "current" may no longer be valid.   MMS depends on this to send properly.
            String operator = mCdmaPhone.mIccRecords.getOperatorNumeric();

            mCdmaPhone.updateCurrentCarrierInProvider(operator);

            // TODO: It'd be nice to only do this if the changed entrie(s)
            // match the current operator.
            createAllApnList();
            if (IsPreferredApnChanged()) {
           Log.i(LOG_TAG, " test onApnChanged IsPreferredApnChanged() in");

               if (mState != State.DISCONNECTING) {
                  Log.i(LOG_TAG, "test onApnChanged mState != State.DISCONNECTING");
                  cleanUpConnection(isConnected, Phone.REASON_APN_CHANGED);
                  if (!isConnected) {
                      Log.i(LOG_TAG, " test onApnChanged !isConnected");
                      // reset reconnect timer
                      //mRetryMgr.resetRetryCount();
                      //mReregisterOnReconnectFailure = false;
                      log("CDMA APN onApnChanged    ");
                      trySetupData(Phone.REASON_APN_CHANGED);
                 }
               }
           }
      }
```

### 6.3.3 The modification of file named RuimRecord.java

Modify the function named onAllRecordsLoaded():

```
Proctected void onAllRecordsLoaded()   {
        Log.d(LOG_TAG, "RuimRecords: record load complete");
        //Further records that can be inserted are Operator /OEM dependent
  - String operator = getRUIMOperatorNumeric();
  -SystemProperties.set(PROPERTY_ICC_OPERATOR_NUMERIC, operator);
  + String operator = getRUIMOperatorNumeric();
  +phone.setSystemProperty(PROPERTY_ICC_OPRATOR_NUMERIC,operator);
  - setSystemProperty(PROPERTY_ICC_OPRATOR_NUMERIC,operator);
}
```

Add an new function named getOperatorNumeric():

```
+Public String getOperatorNumeric()   {
+        if (mImsi == null)   {
+                return null;
+        }
+if (mncLength != UNINITIALIZED && mncLength != UNKNOWN )    {
+return mImsi.substring(0,3 + mncLength);
+}
int mmc = Integer.parseInt(mImsi.substring(0,3));
return mImsi.substring(0, 3 + MccTable.smallestDigitMccForMnc(mcc));
}
```

### 6.3.4 The modification of file named DataConnectionTracker.java

Modify the function named onEnableApn():

```
enabledCount--;
didDisable = true;
}
}
if( didDisable && enabledCount == 0)   {
onCleanUpConnection(true, apnId, Phone.REASON_DATA_DISABLED);
notifyApnIdDisconnected(Phone. REASON_DATA_DISABLED, apnId);
+ }
+    else if (dataEnabled[APN_DEFAULT_ID] ==true
-     if (dataEnabled[APN_DEFAULT_ID] ==true
&& ! isApnTypeActive(Phone.APN_TYPE_DEFAULT))   {
mRequestApnType = phone.APN_TYPE_DEFAULT;
onEnableNewApn();
-    }
}
```

### 6.3.5 The modification of xml files

The modification of xml files are the same as other Android version.

## 6.4 Compile, Download and Run

Before compiling the system, make sure that the modifications of App and Framework layer were compiled. Download the system file, and load the RIL library file of CDMA branch. Using the CDMA modem with SIM card, you can find the corresponding content in the setting menu. The existing APN information can be displayed as the figure 6.4.1 shows.

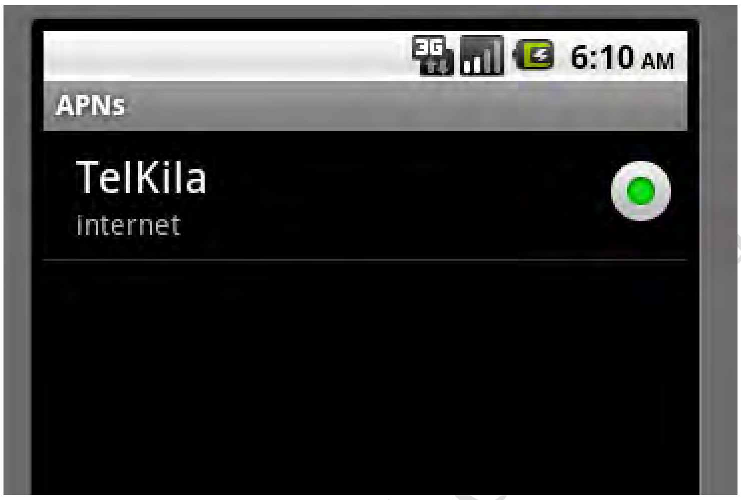


Figure 6.4.1 APN display interface

You can add a new APN, the interface of adding APN as follows:

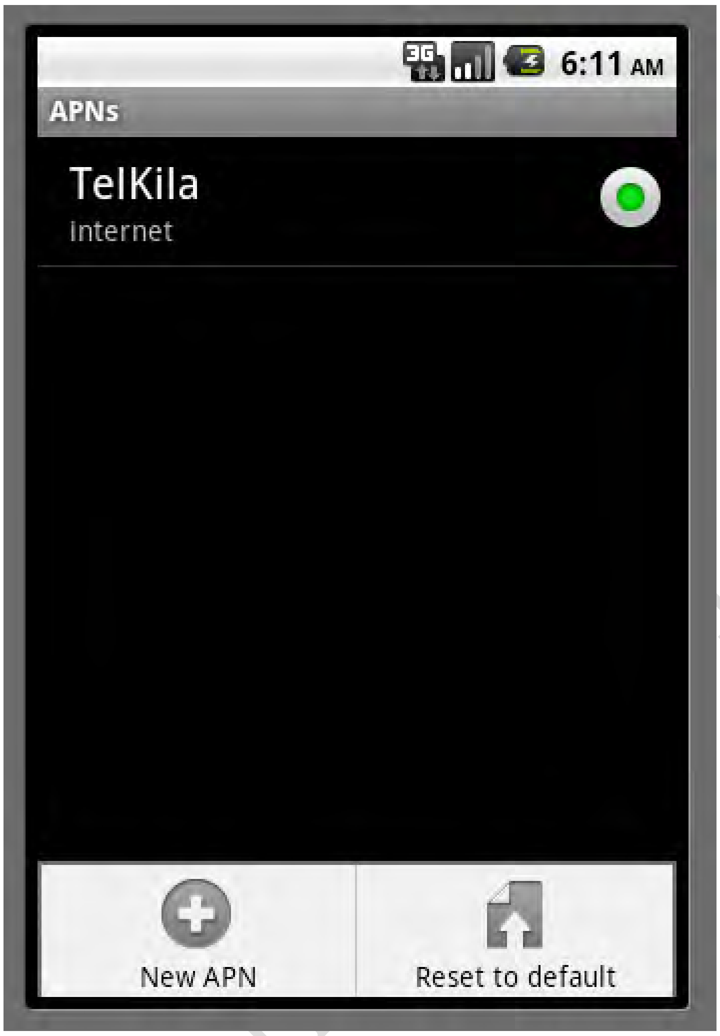
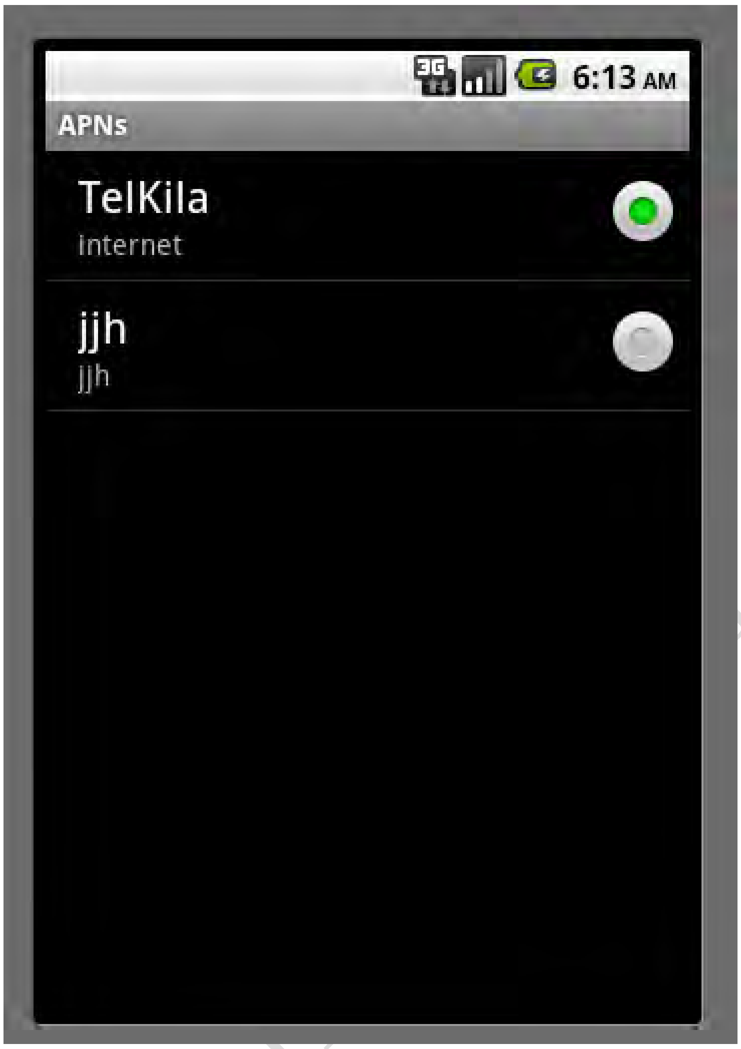


Figure 6.4.2 New APN display interface

The new APN can be saved in the APN list. The interface that you can select and edit APN info is as follows.



TelKila
internet

jjh
jjh

Figure 6.4.2 Select and edit APN interface

# 7 The modification to add call waiting and call forward in CDMA branch

## 7.1 Indroduce

In common Android source code of CDMA branch, there is no function of call waiting and call forward.

In the source code of Android2.2 and Android2.3, at the setting menu of CDMA branch, the settings->Call settings list will not display the “Call forwarding”, “Call waiting” part of the contents are as the figures 7.1.1 and 7.1.2 shows, and will not display the interface as the figure 7.1.3 shows. In order to realize the call waiting and call forward functions, you must add some dispose in the layers of APP and Framework.

In the Android4.0 source code, "Call settings" interface is different from android2.x. In dialing interface click on the menu button, display “call settings” interface, there is no “call waiting" and "call forward" functions in android 4.0. This section mainly introduces the source code added in android2.x and android4.x to achieve the functions.

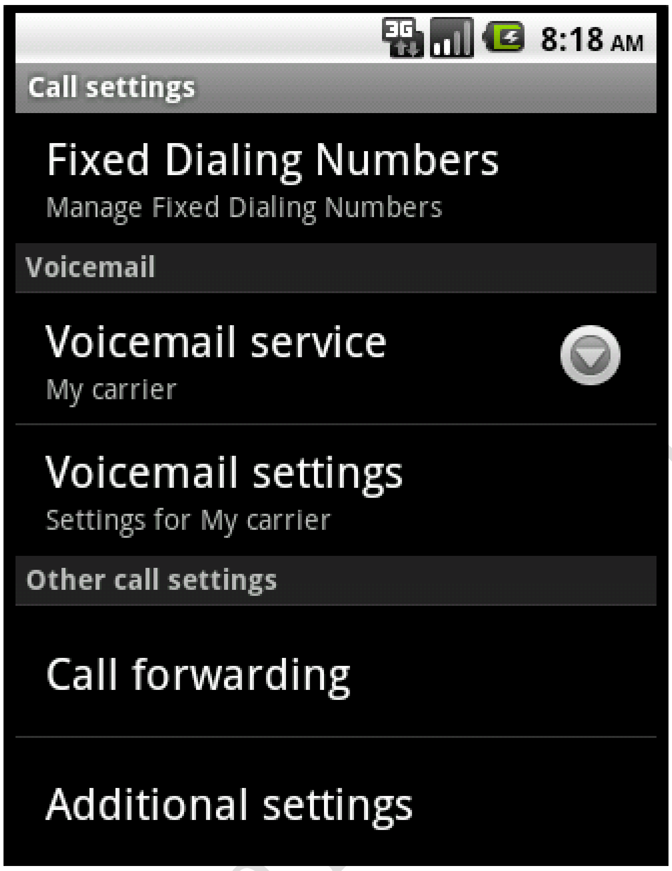

8:18 AM
Call settings
Fixed Dialing Numbers
Manage Fixed Dialing Numbers
Voicemail
Voicemail service
My carrier
Voicemail settings
Settings for My carrier
Other call settings
Call forwarding
Additional settings

Figure 7.1.1 The call setting page

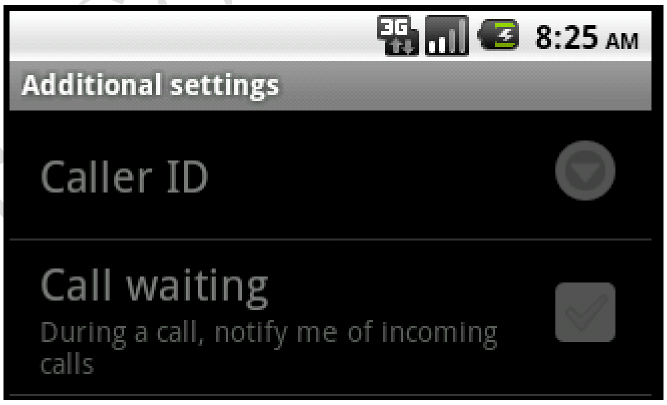
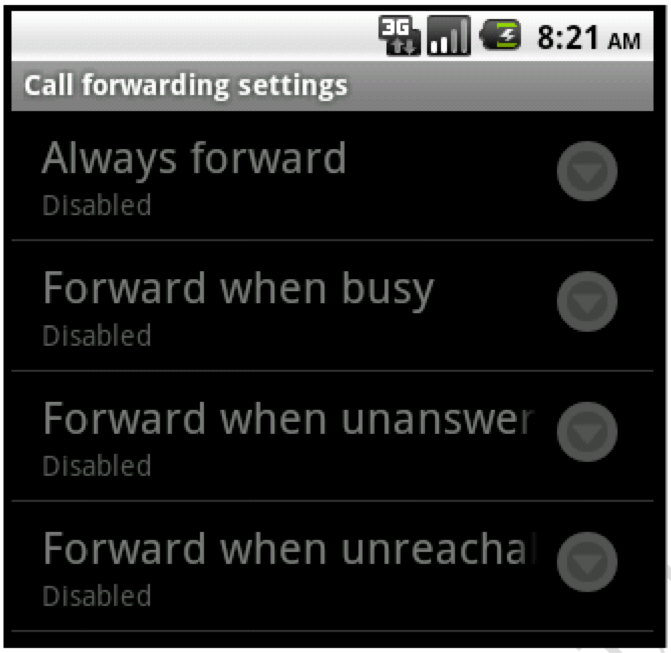

8:25 AM
Additional settings
Caller ID
Call waiting
During a call, notify me of incoming calls

Figure 7.1.2 The Additional settings page

The call forward options list is as follow:



8:21 AM

Call forwarding settings

Always forward
Disabled

Forward when busy
Disabled

Forward when unanswer
Disabled

Forward when unreachal
Disabled

Figure 7.1.3 The call forward setting page

The CDMA call forward and call waiting functions are initiated by dial mode. The state is recorded in network side. The dial-up number is composed of call forward (call waiting) code and phone number. The call forward (call waiting) code is decided by operators. Different types of call forward are corresponded with the different call forward code. For example, in China Telecom, the started code of call forward when busy is 90, the stoped code of call forward when busy is 900; and the started code of call forward when unreachable is 68, the stoped code of call forward when unreachable is 680.The destination telephone number for call forward is edit by user own, but in the call waiting function it doesn’t need a destination telephone number, there is only the startup code of 74 and shutdown code of 740. So you just need modify at the layers of APP and Framework. Now we will introduce the relevant modification. At the APP layer the modify content located in the directory of /packages/apps/Phone/, at the Framework layer, modify the file of CommandsInterface.java in the directory of \frameworks\base\telephony\java\com\android\internal\telephony\.

## 7.2 Files Modification

### 7.2.1 Add new xml file

#### 7.2.1.1Add new file cdma_callforward_options.xml

In order to add call forward interface in CDMA branch, we need to add the file named *cdma_callforward_options.xml* in the directory of \packages\apps\Phone\res\xml\ to achieve that. The content in this file was the same as the file named callforward_options.xml, but at the end of file, we need to add the function of canceling call forward.The code to achieve the function of adding the new page content is as follows (The modification of code in Android2.2, 2.3, 4.0 is the same):

```
        android:autoText="false"/>
```

```
+   <!-- See note on com.android.phone.EditPreference above -->
+       <!--meiqin add for canel callforward-->
+     <com.android.phone.CanelCallForwardPreference
+           android:key="button_cfca_key"
+           android:title="@string/labelCFCA"
+           android:persistent="false"
+           android:dialogTitle="@string/labelCFCA"
+           android:dialogMessage="@string/messageCFCA"
+           phone:confirmMode="activation"
+           phone:serviceClass="voice"
+           phone:reason="cancel_all"
+           android:singleLine="true"
+           android:autoText="false"/>
</PreferenceScreen>
```

### 7.2.2 Modify the files of xml

#### 7.2.2.1 Modify the file: cdma_call_options.xml

In the file of ..\packages\apps\Phone\res\xml\cdma_call_options.xml, there is only "Voice Privacy" in content, add the content as follows in order to achieve the functions of call forward and call waiting (The modification of code is only used in source code of Android 2.2 and Android 2.3).

```
            android:title="@string/additional_cdma_call_settings">
+     <PreferenceScreen
+           android:key="button_cf_expand_key"
+           android:title="@string/labelCF"
+           android:persistent="false">

+           <intent android:action="android.intent.action.MAIN"
+                 android:targetPackage="com.android.phone"
                +android:targetClass="com.android.phone.CdmaCallForwardOptions"/>
+     </PreferenceScreen>

+     <com.android.phone.CdmaCallWaiting
+           android:key="button_cw_key"
+           android:title="@string/labelCW"
+           android:persistent="false"
+           android:summaryOn="@string/sum_cw_enabled"
+           android:summaryOff="@string/sum_cw_disabled"/>

      <com.android.phone.CdmaVoicePrivacyCheckBoxPreference
```

#### 7.2.2.2 Modify the file: cdma_call_privacy.xml

This modification of the code is used in Android4.0 source code only. In Android4.0 the file named cdma_call_privacy.xml is the same as cdma_call_options.xml in android2.2. So add the content as follow in the file of ..\packages\apps\Phone\res\xml\cdma_call_privacy.xml.

```
          android:title="@string/additional_cdma_call_settings">
+       <PreferenceScreen
+           android:key="button_cf_expand_key"
+           android:title="@string/labelCF"
+           android:persistent="false">

+           <intent android:action="android.intent.action.MAIN"
+               android:targetPackage="com.android.phone"
              +android:targetClass="com.android.phone.CdmaCallForwardOptions"/>
+       </PreferenceScreen>

+       <com.android.phone.CdmaCallWaiting
+           android:key="button_cw_key"
+           android:title="@string/labelCW"
+           android:persistent="false"
+           android:summaryOn="@string/sum_cw_enabled"
+           android:summaryOff="@string/sum_cw_disabled"/>

      <com.android.phone.CdmaVoicePrivacyCheckBoxPreference
```

#### 7.2.2.3 Modify the file: pref_dialog_ editphonenumber.xml

In oreder to add the pages of call forward and call waiting dialog, we need to modify the code as follows in the file named \packages\apps\Phone\res\layout\pref_dialog_editphonenumber.xml (The modification of code in Android2.2, Android2.3 and Android4.0 is the same):

```
      android:orientation="vertical">
+ <ScrollView
+       android:layout_width="fill_parent"
+       android:layout_height="fill_parent"
+       >
+ <LinearLayout
+       android:layout_width="fill_parent"
+       android:layout_height="fill_parent"
+       android:orientation="vertical">
       <TextView android:id="@+android:id/message"
         ……
          android:paddingRight="10dip"/>

+         <RadioGroup android:id="@+id/radioGroup"
+             android:layout_width="wrap_content"
+             android:layout_height="wrap_content">
+             <RadioButton android:id="@+id/radioButtonopen"
+                android:layout_width="wrap_content"
+                android:layout_height="wrap_content"
+                android:text="@string/enabled"/>
+            <RadioButton android:id="@+id/radioButtonclose"
+                android:layout_width="wrap_content"
+                android:layout_height="wrap_content"
+                android:text="@string/ban" />
+        </RadioGroup>
      <LinearLayout
      ……
      </LinearLayout>
+ </LinearLayout>
+ </ScrollView>
</LinearLayout>
```

### 7.2.2.4 Modify the file: attrs.xml

In the file of \ packages\apps\Phone\res\values\attrs.xml, add the property of canceling the call forward. Modify the code as follows (In Android2.2, Android2.3 and Android4.0 the modification is the same).

```
                  <enum name="not_reachable" value="3" />
                  <!-- cancel_all -->
+                  <enum name="cancel_all" value="6" />
            </attr>
```

### 7.2.2.5 Modify the file: strings.xml

In file \packages\apps\Phone\res\values\strings.xml, add the text about call waiting and call forward (In Android2.2, Android2.3 and Android4.0 the modification is the same).

```
+      <string name="cnp">caller number display</string>
+      <string name="labelCFCA">Cancel All Forward</string>
+      <string name="messageCFCA">Cancel All Forward</string>

+      <string name="pref_title_bar_cancel_all">Cancel All</string>
+      <string name="dlg_title_cancel_all">Cancel All</string>
+      <string name="dlg_cancel_all_ok">OK</string>
+      <string name="dlg_cancel_all_cancel">cancel</string>
+      <string name="enabled"> Enabled</string>
+      <string name="ban"> Disable</string>
```

### 7.2.2.6 Modify the file: AndroidManifest.xml

In order to add the call forward interface, the content in the file named \packages\apps\Phone\ AndroidManifest.xml should be added as follows (In Android2.2, Android2.3 and Android4.0 the modification is the same).

```
        <activity android:name="CdmaCallOptions"
            android:label="@string/cdma_options">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>

+         <activity android:name="CdmaCallForwardOptions"
+             android:label="@string/labelCF"
+             android:configChanges="orientation|keyboardHidden">
+             <intent-filter>
+                 <action android:name="android.intent.action.MAIN" />
+             </intent-filter>
+         </activity>

        <activity android:name="GsmUmtsCallForwardOptions"
```

### 7.2.3 Add new java file

Now we will introduce the modifications of the java files.

result2.TXT

#### 7.2.3.1 Add the file: CdmaCallForwardOptions.java

Add three files to the directory of \packages\apps\Phone\src\com\android\phone. The file named CdmaCallForwardOptions.java and GsmUmtsCallForwardOptions.java are relatively similar. The differences between them are as follows (In Android2.2, Android2.3 the modification of the code is the same. But there are also somewhat differences from Android4.0 which is written in blue).

```
+public class CdmaCallForwardOptions extends TimeConsumingPreferenceActivity {
+    private static final String LOG_TAG = "CdmaCallForwardOptions";
     private final boolean DBG = (PhoneApp.DBG_LEVEL >= 2);
     private static final String BUTTON_CFNRC_KEY = "button_cfnrc_key";
+    private static final String BUTTON_CFCA_KEY = "button_cfca_key";
     private static final String KEY_TOGGLE = "toggle";
     private CallForwardEditPreference mButtonCFNRy;
     private CallForwardEditPreference mButtonCFNRc;
+    private CanelCallForwardPreference mButtonCFCA;
     private ArrayList<CallForwardEditPreference> mPreferences =
                 new ArrayList<CallForwardEditPreference> ();
     private int mInitIndex= 0;
```

```
-     private boolean mFirstResume;
-       private Bundle mIcicle;
+        private    int modeNo =    1;

-   addPreferencesFromResource(R.xml. callforward_options);
+   addPreferencesFromResource(R.xml.cdma_callforward_options);// Applied to android2.x
+   addPreferencesFromResource(R.xml.cdma_callforward_privacy);// Applied to android4.0
  @Override
mButtonCFNRc = (CallForwardEditPreference) prefSet.findPreference(BUTTON_CFNRC_KEY);
+mButtonCFCA = (CanelCallForwardPreference)
prefSet.findPreference(BUTTON_CFCA_KEY);
mButtonCFU.setParentActivity(this,  mButtonCFU.reason);-                // we wait to do the
initialization until onResume so that the
-             // TimeConsumingPreferenceActivity dialog can display as it
-             // relies on onResume / onPause to maintain its foreground state.
-
-             mFirstResume = true;
-             mIcicle = icicle;
-       }
-
-       @Override
-       public void onResume() {
-             super.onResume();
-
-             if (mFirstResume) {
-                    if (mIcicle == null) {
-                          if (DBG) Log.d(LOG_TAG, "start to init ");
-                          mPreferences.get(mInitIndex).init(this, false);
-                    } else {
-                          mInitIndex = mPreferences.size();
-
-                          for (CallForwardEditPreference pref : mPreferences) {
-                                 Bundle bundle = mIcicle.getParcelable(pref.getKey());
-                                 pref.setToggled(bundle.getBoolean(KEY_TOGGLE));
-                                 CallForwardInfo cf = new CallForwardInfo();
-                                 cf.number = bundle.getString(KEY_NUMBER);
-                                 cf.status = bundle.getInt(KEY_STATUS);
-                                 pref.handleCallForwardResult(cf);
-                                 pref.init(this, true);
-                          }
-                    }
-                    mFirstResume = false;
-                    mIcicle=null;
-             }
-       }
-
-       @Override
-       protected void onSaveInstanceState(Bundle outState) {
-               super.onSaveInstanceState(outState);
-
-               for (CallForwardEditPreference pref : mPreferences) {
-                       Bundle bundle = new Bundle();
-                       bundle.putBoolean(KEY_TOGGLE, pref.isToggled());
-                       if (pref.callForwardInfo != null) {
-                               bundle.putString(KEY_NUMBER, pref.callForwardInfo.number);
-                               bundle.putInt(KEY_STATUS, pref.callForwardInfo.status);
-                       }
-                       outState.putParcelable(pref.getKey(), bundle);
-               }
-       }
-
-       @Override
-       public void onFinished(Preference preference, boolean reading) {
-               if (mInitIndex < mPreferences.size()-1 && !isFinishing()) {
-                       mInitIndex++;
-                       mPreferences.get(mInitIndex).init(this, false);
-               }-
-               super.onFinished(preference, reading);
-       }
```

#### 7.2.3.2 Add the file: CanelCallForwardPreference.java

The added file named CanelCallForwardPreference.java is very similar to the file of CallForwardEditPreference.java.The modification of this file is as follows (In Android2.2, Android2.3 and Android4.0 the modification is the same).

```
import static com.android.phone.TimeConsumingPreferenceActivity.EXCEPTION_ERROR;
import static com.android.phone.TimeConsumingPreferenceActivity.RESPONSE_ERROR;

-   public class CallForwardEditPreference extends EditPhoneNumberPreference {
-   private static final String LOG_TAG = "CallForwardEditPreference";
+    public class CanelCallForwardPreference extends DialogPreference {
+      private static final String LOG_TAG = "CanelCallForwardPreference";
private static final boolean DBG = (PhoneApp.DBG_LEVEL >= 2);

        Phone phone;
+        Context mContext;
        TimeConsumingPreferenceListener tcpListener;
```

```
-       public CallForwardEditPreference(Context context, AttributeSet attrs) {
+       public CanelCallForwardPreference(Context context, AttributeSet attrs) {
              super(context, attrs);
+             mContext = context;

-             phone = PhoneFactory.getDefaultPhone();
-             mSummaryOnTemplate = this.getSummaryOn();

-       public CallForwardEditPreference(Context context) {
+       public CanelCallForwardPreference(Context context) {

   if (!skipReading) {
                    phone.getCallForwardingOption(reason,
-                               mHandler.obtainMessage(MyHandler.MESSAGE_GET_CF,
-                                           // unused in this case
-                                           CommandsInterface.CF_ACTION_DISABLE,
-                                           MyHandler.MESSAGE_GET_CF, null));
+                               mHandler.obtainMessage(MyHandler.MESSAGE_GET_CF, reason,
+                               MyHandler.MESSAGE_GET_CF, null));
                    if (tcpListener != null) {
                          tcpListener.onStarted(this, true);

            }
         }

+       @Override
+       protected void onBindDialogView(View view) {
+             super.onBindDialogView(view);
+             this.mButtonClicked = DialogInterface.BUTTON2;
+       }
```

In the function named protected void onDialogClosed(boolean positiveResult):

```
        protected void onDialogClosed(boolean positiveResult) {
              super.onDialogClosed(positiveResult);

-        if (DBG) Log.d(LOG_TAG, "mButtonClicked=" + mButtonClicked
+       Log.d(LOG_TAG, "    onDialogClosed" );
+       if (DBG) Log.d(LOG_TAG, " mButtonClicked=" + mButtonClicked
                        + ", positiveResult=" + positiveResult);
        if (this.mButtonClicked != DialogInterface.BUTTON_NEGATIVE) {
-         int action = (isToggled() || (mButtonClicked == DialogInterface.BUTTON_POSITIVE)) ?
+                    int action = (mButtonClicked == DialogInterface.BUTTON_POSITIVE) ?
                               CommandsInterface.CF_ACTION_REGISTRATION :
                               CommandsInterface.CF_ACTION_DISABLE;
```

```
                int time = (reason != CommandsInterface.CF_REASON_NO_REPLY) ? 0 : 20;
-               final String number = getPhoneNumber();

                if (DBG) Log.d(LOG_TAG, "callForwardInfo=" + callForwardInfo);
+

-               if (action == CommandsInterface.CF_ACTION_REGISTRATION
-                       && callForwardInfo != null
-                       && callForwardInfo.status == 1
-                       && number.equals(callForwardInfo.number)) {
-                   // no change, do nothing
-                   if (DBG) Log.d(LOG_TAG, "no change, do nothing");
-               } else {
-                   // set to network
-                   if (DBG) Log.d(LOG_TAG, "reason=" + reason + ", action=" + action
-                           + ", number=" + number);
-
-                   // Display no forwarding number while we're waiting for
-                   // confirmation
-                   setSummaryOn("");
-
-                   // the interface of Phone.setCallForwardingOption has error:
-                   // should be action, reason...
-                   phone.setCallForwardingOption(action,
-                           reason,
-                           number,
-                           time,
-                           mHandler.obtainMessage(MyHandler.MESSAGE_SET_CF,
-                                   action,
-                                   MyHandler.MESSAGE_SET_CF));
-
+           /*ZTE _CDMAAPN_001 begin*/
+
+           Log.d(LOG_TAG, " onDialogClosed" + reason + ", action=" + action + ", number=" );
+                   cdmaSetCallForwardingOption(action, reason, null);
+                   Log.d(LOG_TAG, " CdmaCallForwardOptions=" );
                    if (tcpListener != null) {
                        tcpListener.onStarted(this, false);
                    }
-               }
            }
+           /*ZTE _CDMAAPN_001 end*/
+             this.mButtonClicked = DialogInterface.BUTTON2;
        }
```

In the function named handleCallForwardResult, the modification is as follows.

```
        void handleCallForwardResult(CallForwardInfo cf) {
               callForwardInfo = cf;
               if (DBG) Log.d(LOG_TAG, "handleGetCFResponse done, callForwardInfo=" +
             callForwardInfo);
-              setToggled(callForwardInfo.status == 1);
-              setPhoneNumber(callForwardInfo.number);
        }
-
```

In the function named updateSummaryText(), the modification is as follows.

```
  +/**
        private void updateSummaryText() {
               if (isToggled()) {
                      CharSequence summaryOn;
@@ -143,11 +174,72 @@
               }


        }
+**/
+       //modified by liud
+       private void cdmaSetCallForwardingOption(int action, int reason, String number) {
+     String dialnumber = null;
+     Log.d("cdmasetcall", "cdmaSetCallForwardingOption: action, reason"+action+reason);
+     if(action == CommandsInterface.CF_ACTION_REGISTRATION){
+            switch(reason){
+            case CommandsInterface.CF_REASON_UNCONDITIONAL:
+                   dialnumber = "*72"+number;
+                   break;
+            case CommandsInterface.CF_REASON_BUSY:
+                   dialnumber = "*90"+number;
+                   break;
+            case CommandsInterface.CF_REASON_NO_REPLY:
+                   dialnumber = "*92"+number;
+                   break;
+            case CommandsInterface.CF_REASON_NOT_REACHABLE:
+                   dialnumber = "*68"+number;
+                   break;
+            case CommandsInterface.CF_REASON_CANCEL_ALL:
+                   dialnumber = "*730";
+                   break;
+            default:
+                   if (DBG)
```

```
   Log.d(LOG_TAG, "cdmaSetCallForwardingOption CF_ACTION_REGISTRATION : nothing to
do");
+                 break;
+             }
+         }
+     else if(action == CommandsInterface.CF_ACTION_DISABLE){
+             switch(reason){

-       // Message protocol:
-       // what: get vs. set
-       // arg1: action -- register vs. disable
-       // arg2: get vs. set for the preceding request
+             case CommandsInterface.CF_REASON_UNCONDITIONAL:
+                 dialnumber = "*720";
+                 break;
+             case CommandsInterface.CF_REASON_BUSY:
+                 dialnumber = "*900";
+                 break;
+             case CommandsInterface.CF_REASON_NO_REPLY:
+                 dialnumber = "*920";
+                 break;
+             case CommandsInterface.CF_REASON_NOT_REACHABLE:
+                 dialnumber = "*680";
+                 break;
+             default:
+                 if (DBG)
 Log.d(LOG_TAG, "cdmaSetCallForwardingOption CF_ACTION_DISABLE : nothing to do");
+                 break;
+             }
+         }
+     else{
+             if (DBG) Log.d(LOG_TAG, "cdmaSetCallForwardingOption: invalide check");
+     }
+         if(dialnumber == null) return;
+     if (DBG) Log.d(LOG_TAG, "cdmaSetCallForwardingOption: dialnumber = "+dialnumber);
+             Intent intent = new Intent(Intent.ACTION_CALL_PRIVILEGED,
+                         Uri.fromParts("tel", dialnumber, null));
+             //Log.d("ZTE", "mode" +mode);
+             //intent.putExtra("com.android.phone.DialingModeNo",mode);
+             intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
+             mContext.startActivity(intent);
+
+         }
+         //endded
```

```
        private class MyHandler extends Handler {
                private static final int MESSAGE_GET_CF = 0;
                private static final int MESSAGE_SET_CF = 1;
```

In the function named handleGetCFResponse(Message msg), the modification is as follows.

```
if (msg.arg2 == MESSAGE_SET_CF) {
- tcpListener.onFinished(CallForwardEditPreference.this, false);
+tcpListener.onFinished(CanelCallForwardPreference.this, false);
                          } else {
- tcpListener.onFinished(CallForwardEditPreference.this, true);
+ tcpListener.onFinished(CanelCallForwardPreference.this, true);
                          }

callForwardInfo = null;
  if (ar.exception != null) {
- if (DBG) Log.d(LOG_TAG, "handleGetCFResponse: ar.exception=" + ar.exception);
+    if (DBG) Log.d(LOG_TAG, " handleGetCFResponse: ar.exception=" + ar.exception);
                                  setEnabled(false);
-tcpListener.onError(CallForwardEditPreference.this, EXCEPTION_ERROR);
+tcpListener.onError(CanelCallForwardPreference.this, EXCEPTION_ERROR);
                          } else {
                                  if (ar.userObj instanceof Throwable) {
-tcpListener.onError(CallForwardEditPreference.this, RESPONSE_ERROR);
+tcpListener.onError(CanelCallForwardPreference.this, RESPONSE_ERROR);
                                  }
                                  CallForwardInfo cfInfoArray[] = (CallForwardInfo[]) ar.result;
                                  if (cfInfoArray.length == 0) {
-if (DBG) Log.d(LOG_TAG, "handleGetCFResponse: cfInfoArray.length==0");
+if (DBG) Log.d(LOG_TAG, " handleGetCFResponse: cfInfoArray.length==0");
                                          setEnabled(false);
-tcpListener.onError(CallForwardEditPreference.this, RESPONSE_ERROR);
+tcpListener.onError(CanelCallForwardPreference.this, RESPONSE_ERROR);
                                  } else {
                                          for (int i = 0, length = cfInfoArray.length; i < length; i++) {
-cfInfoArray[" + i + "]="
+if (DBG) Log.d(LOG_TAG, " handleGetCFResponse, cfInfoArray[" + i + "]="
                                                          + cfInfoArray[i]);
                                                  if ((mServiceClass & cfInfoArray[i].serviceClass) != 0) {
                                                          // corresponding class
-CallForwardInfo info = cfInfoArray[i];
-handleCallForwardResult(info);
-
- // Show an alert if we got a success response but
- // with unexpected values.
- // Currently only handle the fail-to-disable case
- // since we haven't observed fail-to-enable.
-                     if    (msg.arg2    ==    MESSAGE_SET_CF    &&    msg.arg1    ==
CommandsInterface.CF_ACTION_DISABLE &&- info.status == 1) {
- CharSequence s;
- switch (reason) {
- case CommandsInterface.CF_REASON_BUSY:
- s = getContext().getText(R.string.disable_cfb_forbidden);
- break;
- case CommandsInterface.CF_REASON_NO_REPLY:
- getContext().getText(R.string.disable_cfnry_forbidden);
- break;
- default: // not reachable
- s = getContext().getText(R.string.disable_cfnrc_forbidden);
- }
- AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
- builder.setNeutralButton(R.string.close_dialog, null);
-builder.setTitle(getContext().getText(R.string.error_updating_title));
- builder.setMessage(s);
- builder.setCancelable(true);
- builder.create().show();
-       }
+ handleCallForwardResult(cfInfoArray[i]);
      }
    }
  }

-   updateSummaryText();
```

In the function named handleSetCFResponse , the modification is as follows.

```
     phone.getCallForwardingOption(reason,
–    obtainMessage(MESSAGE_GET_CF, msg.arg1, MESSAGE_SET_CF, ar.exception));
+      obtainMessage(MESSAGE_GET_CF, reason, MESSAGE_SET_CF, ar.exception));
```

#### 7.2.3.3 Add the file: CdmaCallWaiting.java

Add the file named CdmaCallWaiting.java to \packages\apps\Phone\src\com\android\phone\, the content as follows.

```
package com.android.phone;

import static com.android.phone.TimeConsumingPreferenceActivity.EXCEPTION_ERROR;
import static com.android.phone.TimeConsumingPreferenceActivity.RESPONSE_ERROR;
//importstatic com.android.phone.TimeConsumingPreferenceActivity.FDN_BLOCKED_ERROR;
```

```
import com.android.internal.telephony.Phone;
import com.android.internal.telephony.PhoneFactory;
import android.content.DialogInterface;
import android.net.Uri;
import android.content.Intent;
import android.content.Context;
import android.os.AsyncResult;
import android.os.Handler;
import android.os.Message;
import android.preference.CheckBoxPreference;
import android.util.AttributeSet;
import android.util.Log;
import com.android.internal.telephony.CommandException;
import android.content.res.TypedArray;
import com.android.internal.telephony.CommandsInterface;
import android.view.View;

public class CdmaCallWaiting extends EditPhoneNumberPreference {
    private static final String LOG_TAG = "CallWaitingCheckBoxPreference";
    private final boolean DBG = (PhoneApp.DBG_LEVEL >= 2);
    Phone phone;

    private int mode = 1;
    private int mServiceClass;
    int reason;
    private int mButtonClicked;

    TimeConsumingPreferenceListener tcpListener;
    Context mContext;
    public CdmaCallWaiting(Context context, AttributeSet attrs) {
        super(context, attrs);
        Log.d("mq", "CallForwardEditPreference");
        mContext = context;
        //phone = PhoneFactory.getPhoneWithModeNo(1); jh712
        phone = PhoneFactory.getDefaultPhone();
        //mSummaryOnTemplate = this.getSummaryOn();

        TypedArray a = context.obtainStyledAttributes(attrs,
           R.styleable.CallForwardEditPreference, 0, R.style.EditPhoneNumberPreference);
        mServiceClass = a.getInt(R.styleable.CallForwardEditPreference_serviceClass,
        CommandsInterface.SERVICE_CLASS_VOICE);
        reason = a.getInt(R.styleable.CallForwardEditPreference_reason,
        CommandsInterface.CF_REASON_UNCONDITIONAL);
        a.recycle();

      if (DBG) Log.d(LOG_TAG, "mServiceClass=" + mServiceClass + ", reason=" + reason);
    }

    public CdmaCallWaiting(Context context) {
        this(context, null);
    }

    public void setMode() {
         Log.d(LOG_TAG, "mq~~~~setmode"+mode);
         setModeNo(0);      //for cdma callwaiting
    }
  @Override
    public void onClick(DialogInterface dialog, int which) {
        super.onClick(dialog, which);
        mButtonClicked = which;
         Log.d(LOG_TAG, "mqnew~~~~onclick!" );
    }


    @Override
    protected void onBindDialogView(View view) {
            this.mButtonClicked = DialogInterface.BUTTON2;
            userChoice = 1;
            setModeNo(0);    //ZTE_JJH_CDMAAPN_001

        super.onBindDialogView(view);
    }
      @Override
    protected void onDialogClosed(boolean positiveResult) {
        super.onDialogClosed(positiveResult);
        if (DBG) Log.d(LOG_TAG, "mButtonClicked=" + mButtonClicked
                + ", positiveResult=" + positiveResult);
        if (this.mButtonClicked != DialogInterface.BUTTON_NEGATIVE) {
  // int action    = (isToggled() || (mButtonClicked == DialogInterface.BUTTON_POSITIVE)) ?
                         //      CommandsInterface.CF_ACTION_REGISTRATION :
                          //    CommandsInterface.CF_ACTION_DISABLE;
    int action ;

    boolean choice = isToggled() || (mButtonClicked == DialogInterface.BUTTON_POSITIVE);

                    if(choice == true&&userChoice == 1){
                     action = CommandsInterface.CF_ACTION_REGISTRATION;
```

```
                    if (DBG) Log.d("mq", "userChoice1="+userChoice);
                   }else if(choice == true&&userChoice == 2){
                   action = CommandsInterface.CF_ACTION_DISABLE;
                    if (DBG) Log.d("mq", "userChoice2="+userChoice);
                   }else{
                      return;
                   }

              int time = (reason != CommandsInterface.CF_REASON_NO_REPLY) ? 0 : 20;
             // final String number = getPhoneNumber();

                  cdmaSetCallWaiting(action);
                    Log.d(LOG_TAG, "mq~~~~cdmaSetCallWaiting()" );
                   if (tcpListener != null) {
                        tcpListener.onStarted(this, false);
                   }

          }
     }
     private void cdmaSetCallWaiting(int action) {
     if (DBG) Log.d("mq", "action="+action);
     final String dialnumber;
     if(action == CommandsInterface.CF_ACTION_REGISTRATION){
         if (DBG) Log.d("mq", "action= CF_ACTION_REGISTRATION");
         dialnumber = "*74";
         }
     else {
         if (DBG) Log.d("mq", "action= CF_ACTION_DISABLE");
         dialnumber = "*740";
         }
            Log.d("mq", "cdmaSetCallWaiting: dialnumber = "+dialnumber);
          Intent intent = new Intent(Intent.ACTION_CALL_PRIVILEGED,
                   Uri.fromParts("tel", dialnumber, null));

          intent.putExtra("com.android.phone.DialingModeNo",mode);
          /**mq add for CRDB00520745 end **/
          intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
          mContext.startActivity(intent);

     }
     //ended

}
```

### 7.2.4 The Modification of java Files

#### 7.2.4.1 Modify the file: CallForwardEditPreference.java

In the file named \packages\apps\Phone\src\com\android\phone\CallForwardEditPreference.java , the modification content is as follows(The modification of code in Android2.2, Android2.3 and Android4.0 is the same).

```
          Phone phone;
+         int mode;
+         Context mContext;
```

In the constructor function in the class of CallForwardEditPreference:

```
   super(context, attrs);
+ mContext = context;
```

In the function of protected void onDialogClosed(boolean positiveResult):

```
                 CommandsInterface.CF_ACTION_REGISTRATION :
                 CommandsInterface.CF_ACTION_DISABLE;
+
+boolean choice = isToggled() || (mButtonClicked == DialogInterface.BUTTON_POSITIVE);
+        if(choice == true){
+              if(userChoice ==1 ){
+                  action = CommandsInterface.CF_ACTION_REGISTRATION;
+                  if (DBG) Log.d("mq", "userChoice1="+userChoice);
+              }else if(userChoice == 2){
+                  action = CommandsInterface.CF_ACTION_DISABLE;
+                  if (DBG) Log.d("mq", "userChoice2="+userChoice);
+              }
+     }else{
+            return;
+     }
+
+
   int time = (reason != CommandsInterface.CF_REASON_NO_REPLY) ? 0 : 20;

  if (DBG) Log.d(LOG_TAG, "callForwardInfo=" + callForwardInfo);

-                      if (action == CommandsInterface.CF_ACTION_REGISTRATION
-                      if (action == CommandsInterface.CF_ACTION_REGISTRATION
-                                   && callForwardInfo != null
-                                   && callForwardInfo.status == 1
-                                   && number.equals(callForwardInfo.number)) {
-                                  ……
-                              if(tcpListener != null)    {
```

```
-                               tcpListener.onStarted(this, false);
-                           }
-                       }
+*/
+          /*ZTE _CDMAAPN_001 begin*/
+
+                   cdmaSetCallForwardingOption(action, reason, number);
+                         Log.d(LOG_TAG, " CdmaCallForwardOptions=" );
+                         if (tcpListener != null) {
+                               tcpListener.onStarted(this, false);
+                         }
+          /*ZTE _CDMAAPN_001 end*/
+            }
       }
 }
       void handleCallForwardResult(CallForwardInfo cf) {
```

Add the new function as follows:

```
+      private void cdmaSetCallForwardingOption(int action, int reason, String number) {
+    String dialnumber = null;
+    Log.d("mq", "cdmaSetCallForwardingOption: action, reason"+action+reason);
+    if(action == CommandsInterface.CF_ACTION_REGISTRATION){
+          switch(reason){
+
+          case CommandsInterface.CF_REASON_UNCONDITIONAL:
+                dialnumber = "*72"+number;
+                break;
+          case CommandsInterface.CF_REASON_BUSY:
+                dialnumber = "*90"+number;
+                break;
+          case CommandsInterface.CF_REASON_NO_REPLY:
+                dialnumber = "*92"+number;
+                break;
+          case CommandsInterface.CF_REASON_NOT_REACHABLE:
+                dialnumber = "*68"+number;
+                break;
+          case CommandsInterface.CF_REASON_CANCEL_ALL:
+                dialnumber = "*730";
+                break;
+          default:
+                if      (DBG)      Log.d(LOG_TAG,      "cdmaSetCallForwardingOption
CF_ACTION_REGISTRATION : nothing to do");
+                break;
+          }
+       }
+     else if(action == CommandsInterface.CF_ACTION_DISABLE){
+           switch(reason){
+
+           case CommandsInterface.CF_REASON_UNCONDITIONAL:
+                 dialnumber = "*720";
+                 break;
+           case CommandsInterface.CF_REASON_BUSY:
+                 dialnumber = "*900";
+                 break;
+           case CommandsInterface.CF_REASON_NO_REPLY:
+                 dialnumber = "*920";
+                 break;
+           case CommandsInterface.CF_REASON_NOT_REACHABLE:
+                 dialnumber = "*680";
+                 break;
+           default:
+                 if        (DBG)        Log.d(LOG_TAG,        "cdmaSetCallForwardingOption
CF_ACTION_DISABLE : nothing to do");
+                 break;
+           }
+       }
+     else{
+             if (DBG) Log.d(LOG_TAG, "cdmaSetCallForwardingOption: invalide check");
+       }
+     if (DBG) Log.d(LOG_TAG, "cdmaSetCallForwardingOption: dialnumber = "+dialnumber);
+             Intent intent = new Intent(Intent.ACTION_CALL_PRIVILEGED,
+                          Uri.fromParts("tel", dialnumber, null));
+             /**mq add for CRDB00520745 start **/
+             //intent.putExtra("com.android.phone.DialingModeNo",mode);
+             /**mq add for CRDB00520745 end **/
+             intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
+             mContext.startActivity(intent);
+
+       }

        // Message protocol:
```

#### 7.2.4.2 Modify the file: EditPhoneNumberPreference.java

In the file named EditPhoneNumberPreference.java located in the directory of \packages\apps\ Phone\src\com\android\phone\, the modification content is as follows (The modification of code in Android2.2, Android2.3 and Android4.0 is the same).

```
        private String mPhoneNumber;
        private boolean mChecked;

+       public int userChoice=1;
+       private int mode=1;
+       private RadioGroup radioGroup;

        /**
          * Interface for the dialog closed listener, related to

  public EditPhoneNumberPreference(Context context, AttributeSet attrs) {
                super(context, attrs);

setDialogLayoutResource(R.layout.pref_dialog_editphonenumber);
                //create intent to bring up contact list
                mContactListIntent = new Intent(Intent.ACTION_GET_CONTENT);
```

In the function name onBindView:

```
            protected void onBindView(View view)
                        sum = getSummary();
                    }

-                   if (sum != null) {
+                    Log.d("00000","mode ="+mode);
+                    if ((sum != null)&&(mode!=1)) {
+                        Log.d("00001","mode ="+mode);
                          summaryView.setText(sum);
                          vis = View.VISIBLE;
                    } else {
+                         Log.d("00002","mode ="+mode);
                          vis = View.GONE;
                    }
```

Add the new function as follows:

```
+       public void setModeNo(int modeno){
+                    mode = modeno;
+                    Log.d("ZTE","mode ="+mode);
+
+               }
```

In the function of protected void onBindDialogView(View view):

```
   mContactPickButton = (ImageButton) view.findViewById(R.id.select_contact);
  +
+Log.d("ZTE", "editphone PHONE_TYPE_cdma_callwaiting");
+if(mode ==0){
+ Log.d("ZTE", "editphone PHONE_TYPE_cdma_callwaiting");
+ editText.setVisibility(View.GONE);
+ mContactPickButton.setVisibility(View.GONE);
+ }else{
+
                //setup number entry
                if (editText != null)
                  ……
                            }
                      });
+ }
+      radioGroup = (RadioGroup) view.findViewById(R.id.radioGroup);
+final RadioButton radioButtonOpen = (RadioButton) view.findViewById(R.id.radioButtonopen);
+            final            RadioButton        radioButtonClose        =        (RadioButton)
view.findViewById(R.id.radioButtonclose);
+    radioButtonOpen.setChecked(true);
+    if(mode == 2){
+          Log.d("ZTE", "editphone PHONE_TYPE_GSM");
+          radioGroup.setVisibility(View.GONE);
+          radioButtonOpen.setVisibility(View.GONE);
+          radioButtonClose.setVisibility(View.GONE);
+          }else{
+    radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
+             public void onCheckedChanged(RadioGroup group,int checkedId) {
+             Log.d("mq", "userChoice=1"+checkedId);
+             if(checkedId == radioButtonOpen.getId()){
+                userChoice = 1;
+                Log.d("ZTE", "userChoice=1");
+                }else if(checkedId == radioButtonClose.getId()){
+                userChoice = 2;
+                Log.d("ZTE", "userChoice=2");
+                }
+          }
+       });
+    }
}
```

In the function of onPrepareDialogBuilder(AlertDialog.Builder builder), the modification is as follows:

```
// displayed, since there is no need to hide the edittext
 // field anymore.
```

```
+Log.d("ZTE", "editphone EditPhoneNumberPhone    onPrepareDialogBuilder");
 if (mConfirmationMode == CM_ACTIVATION) {
+ Log.d("ZTE", "editphone EditPhoneNumberPhone    onPrepareDialogBuilder");
        if (mChecked) {
+         Log.d("ZTE", "editphone EditPhoneNumberPhone    onPrepareDialogBuilder");
             builder.setPositiveButton(mChangeNumberText, this);
-            builder.setNeutralButton(mDisableText, this);
+
+            if(mode == 2){                              //ZTE_JJH_CDMAAPN_001
+       Log.d("ZTE", "editphone EditPhoneNumberPhone    onPrepareDialogBuilder");
+                    builder.setNeutralButton(mDisableText, this);
+                    }
        } else {
+         Log.d("ZTE", "editphone EditPhoneNumberPhone onPrepareDialogBuilder");
             builder.setPositiveButton(null, null);
             builder.setNeutralButton(mEnableText, this);
        }
```

At last, the modification of code in the layer of framework is in the file named CommandsInterface.java located in the directory of \frameworks\base\telephony\java\com\android\internal\telephony\ (The modification in Android2.2, Android2.3 and Android4.0 is the same).

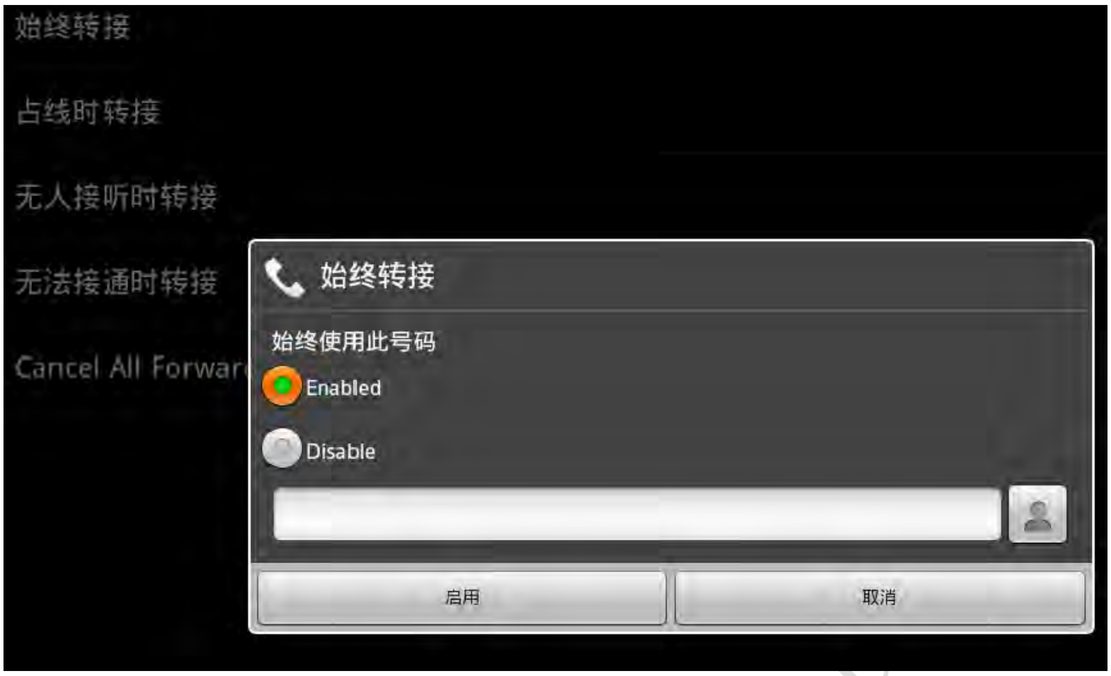
```
     static final int CF_REASON_ALL_CONDITIONAL    = 5;
  +  static final int CF_REASON_CANCEL_ALL    = 6;   //ZTE_JJH_CDMAAPN_001

     // Used for call barring methods below
     static final String CB_FACILITY_BAOC           = "AO";
```

## 7.3 Compile, Download and Run

Before compiling the system, make sure that the modification of the layers of App and Framework are compiled. Download the system file, and load the RIL library file in CDMA branch. Using the CDMA modem with SIM card, then you can find the corresponding content in the setting menu.

When you clicked the call forward button, it will display the dialog box as Figure 7.3.1 shows.



始终转接

占线时转接

无人接听时转接

无法接通时转接

Cancel All Forwar

始终转接

始终使用此号码

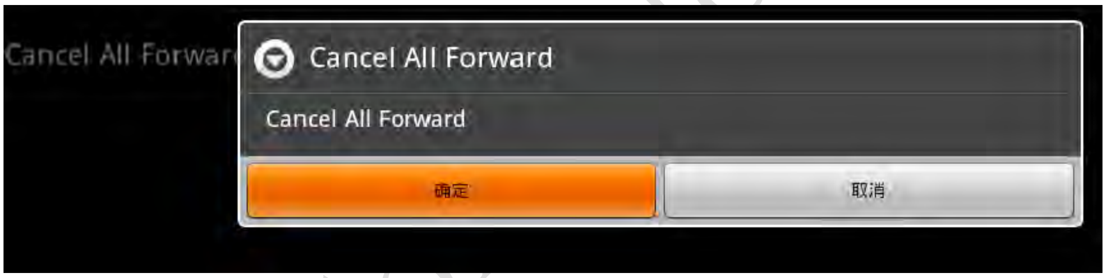
[✓ Enabled]

Disable

启用 取消

Figure 7.3.1 Call forward interface

The cancel call forward dialog is as Figure 7.3.2 show.



Cancel All Forwar

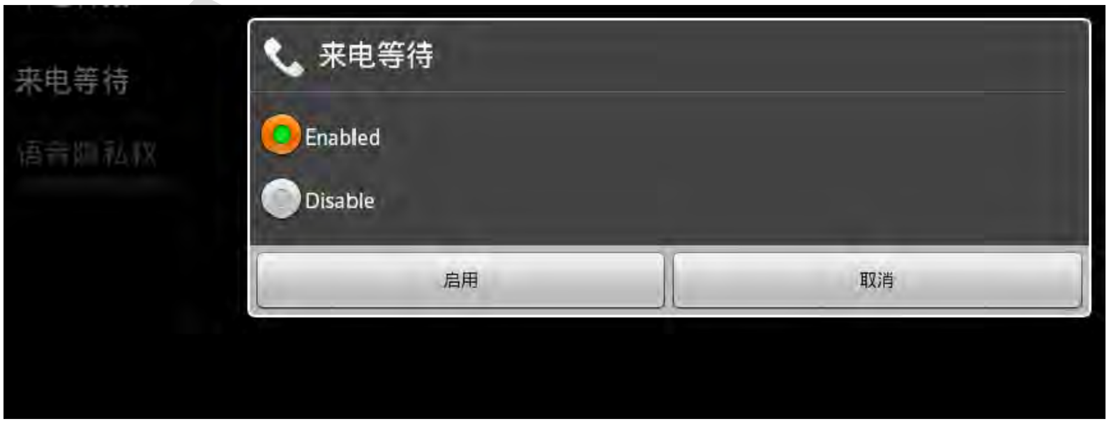
Cancel All Forward

Cancel All Forward

确定 取消

Figure 7.3.2 cancel call forward interface

The dialog of call waiting is as Figure 7.3.3 shows.



来电等待

语音隐私权

来电等待

[✓ Enabled]

Disable

启用 取消

Figure 7.3.3 Call waiting interface

# 8 Voice services

## 8.1 Configuring the voice channel

**Question:** The modem of our company provides several modes of voice channel, different customers can configure according to their own needs.

**Analysis:** In the provided RIL library retains the interface, the customers need to set the property as follows in order to inform the RIL that which voice channel is used currently.

**Solution:** Modify the file named ril.c under the directory of /hardware/ril/rild/, which informs the RIL that the voice channel is used currently by the way to set the property in the system.

```
--- init/rild/rild.c      2011-10-31 14:28:40.054173000 +0800
+++ modified/rild/rild.c  2011-11-21 11:15:07.132922000 +0800
@@ -41,6 +41,13 @@
+#define ZTE_AUDIO_SWITCH    "ril.audio.switch"
+#define ZTE_AUDIO_PCM    "0"
+#define ZTE_AUDIO_LINEIN_LINEOUT_DIFF    "1"
+#define ZTE_AUDIO_MIC_LINEOUT_LEFT_RIGHT    "2"
+#define ZTE_AUDIO_MIC_LINEOUT_DIFF    "3"
static void usage(const char *argv0)
 {
        fprintf(stderr, "Usage: %s -l <ril impl library> [-- <args for impl library>]\n", argv0);
@@ -127,6 +134,7 @@
+        property_set(ZTE_AUDIO_SWITCH, ZTE_AUDIO_MIC_LINEOUT_DIFF, NULL);

        if (rilLibPath == NULL) {
                if ( 0 == property_get(LIB_PATH_PROPERTY, libPath, NULL)) {
```

## 8.2 Call timer

In the Specification of China Telecom's there are two call timer types: call timer and conversation timer. Call duration is started to time when the other's ring rings, and conversation timer is started to time when the other connects the call.

At the present time, in the CDMA networks of China Telecom, we cannot get the signal of phone connecting from network side. Therefore we proposed the scheme of this: we start to time at the time of the other side's status is ringing, and this time is call timer. If the network side sends the signal of phone connecting, clean the time, and start the conversation timer. If there is no the signal of phone connecting, it still displays call timer. To achieve this we need to synchronize the changes at the layer of framework, the modification of cod is as follows:

1 Add an event to identify the event of connecting phone to each other, the concrete code is as

follows:

```
--- init/CallTracker.java  2011-10-31 14:29:11.394174000 +0800
+++ modify/CallTracker.java 2012-02-24 13:35:22.882918000 +0800
@@ -60,6 +60,10 @@
         protected static final int EVENT_CALL_WAITING_INFO_CDMA            = 15;
         protected static final int EVENT_THREE_WAY_DIAL_L2_RESULT_CDMA = 16;

+        //ZTE CALL TIME,begin
+        protected static final int EVENT_CDMA_LINE_CONTROL_INFO = 18;
+        //ZTE CALL TIME,end
+
         protected void pollCallsWhenSafe() {
              needsPoll = true;
```

2 Register the event of EVENT_CDMA_LINE_CONTROL_INFO. When receive this message, clean call timer.

```
--- init/CdmaCallTracker.java 2011-10-31 14:29:11.284173000 +0800
+++ modify/CdmaCallTracker.java       2012-02-24 13:39:36.102918000 +0800
@@ -38,6 +38,12 @@
  import java.util.List;
  +import java.util.Iterator;
+import android.util.Config;

  /**
    * {@hide}
    */
@@ -98,6 +104,11 @@
              cm.registerForNotAvailable(this, EVENT_RADIO_NOT_AVAILABLE, null);
              cm.registerForCallWaitingInfo(this, EVENT_CALL_WAITING_INFO_CDMA, null);
              foregroundCall.setGeneric(false);
+
+        //ZTE CALL TIME,begin
+       cm.registerForLineControlInfo(this, EVENT_CDMA_LINE_CONTROL_INFO, null);
+       //ZTE CALL TIME,end
+
         }

         public void dispose() {
```

```
@@ -105,6 +116,9 @@
            cm.unregisterForOn(this);
            cm.unregisterForNotAvailable(this);
            cm.unregisterForCallWaitingInfo(this);
+     //ZTE CALL TIME,begin
+     cm.unregisterForLineControlInfo(this);
+           //ZTE CALL TIME,end
            for(CdmaConnection c : connections) {
                try {
                    if(c != null) hangup(c);
@@ -1013,6 +1027,48 @@
                        pendingMO = null;
                    }
                break;
+           //ZTE CALL TIME,begin
+           case EVENT_CDMA_LINE_CONTROL_INFO:
+                Log.d(LOG_TAG, "[VOICE]EVENT_CDMA_LINE_CONTROL_INFO");
+
+                ar = (AsyncResult)msg.obj;
+
+                if(ar.exception == null) {
+                  int fgCount = foregroundCall.getConnections().size();
+                  Iterator it = foregroundCall.getConnections().iterator();
+
+                  if((fgCount > 1) && (foregroundCall.getState() == CdmaCall.State.ACTIVE)) {
+                      //note:
+                      //normal there should not be receive FWIM when call was in active state,
+                      //but if indeed received , just ignore it.
+                      Log.d(LOG_TAG, "[VOICE]already have a call in active state, just ignore
msg");
+                  } else {
+
+                                   CdmaInformationRecords.CdmaLineControlInfoRec clcInfoRec =
(CdmaInformationRecords.CdmaLineControlInfoRec) ar.result;
+
+                     if(clcInfoRec.lineCtrlPolarityIncluded == 1) {
+                         //note:because cdma can not recognise different connection
+                               //so update all connections which parent are fgCall,
+                               //but it will lead to three way call logs time unnicety.
+
+                           //in the case first MO call does not be answered,
+                           //then orig second MO call and is answered,
+                           //the first MO call`s time is not precise.
+                               if(fgCount > 1) {
+                                   foregroundCall.setGeneric(true);
+                               }
+
+                           while(it.hasNext()) {
+                                   Log.d(LOG_TAG, "[VOICE]onCdmaLineCtrlInfo");
+                                   CdmaConnection conn = (CdmaConnection) it.next();
+                                   conn.onCdmaLineCtrlInfo();
+                           }
+                         }
+
+                     }
+                   }
+                   break;
+           //ZTE CALL TIME,end
```

3 In the file named *cdmaConnetction.java*, add the function of onCdmaLineCtrlInfo as follows.

```
void onCdmaLineCtrlInfo() {
        Log.d(LOG_TAG, "[VOICE]onCdmaLineCtrlInfo>-");
        connectTime = System.currentTimeMillis();
        connectTimeReal = SystemClock.elapsedRealtime();
        duration = 0;
Log.d(LOG_TAG, "[VOICE]cdmaLineCtrlTime=" + connectTime + ",
cdmaLineCtrlTimeReal=" + connectTimeReal);
        releaseWakeLock();
        Log.d(LOG_TAG, "[VOICE]onCdmaLineCtrlInfo <-");
      }
```

# 9 The operation of SIM card

## 9.1 Save the Chinese contacts in the SIM card

Please consult the chapter of 3.1 in part II

# 10 AT commands sending in the layer of Framework

Please consult the chapter of 6 in part II

# Part IV: Common Fault Processing (Q&A)

## Special Explanation

If you have adapted other manufacturer's modem before, we suggest that you should first remove the adaptation files, and then adapt the ZTE modem.

## 1 Data

### 1.1The unset of APN

APN settings locate in: [settings->Wireless&networks->Mobile network settings->Access point names]. Please inquire the operators or consult phone illumination before setting.

### 1.2 Data connection is not startup

Check the data connection whether it is already started in the menu of [settings->Wireless&networks->Mobile network settings->Data enalbed], if the data conection is not enabled, the data will not be connected.

### 1.3 The format of script named init.gprs-pppd is incorrect

The format of script named init.gprs-pppd is UNIX format, if opening the script using Windows editor, it will be converted to DOS format by system automaticly. After the file converted to DOS format, this script can not run in Android system. Therefore, make sure the script's format is UNIX.

When opening the file in UNIX format using UltraEdit-32, the system will prompt that: Is converted to DOS format or not? At the same time, we can convert DOS format to UNIX format using UltraEdit-32's [file->conversions].

### 1.4 Check the permissions of dial-up scripts

Check the permissions of ip-up-ppp0 and ip-down-ppp0. The two files are located in the directory of /system/etc/ppp in your system. Please confirm whether the two files have executed permissions.

Check the permissions of init.gprs-pppd. This file is located in the directory of /system/etc in your system. Please confirm whether this file has executed permission too.

## 2 SMS

### 2.1 Can not send and receive SMS, Prompt storage space is full

Make sure whether the directory of "/data" could be read and written, and the content write to "/data" directory could be retained when restart after the power down.

## 3 Voice

### 3.1 Unable to make a voice Call in Android4.0.3

This is a bug on some hardware platforms in Android4.0.3. Please consult to the provide manufacturers to solve this.

Voice function can be verificated through making a telephone call after setting the property ro.sf.lcd_density as 160 or larger. Because it will make the screen resolution changes, it is not a Ultimate solution.

## 4 Others

### 4.1Can not load RIL library

If our RIL library can not be loaded by system (have no ZTE LOG print), but if the system's default RIL library could be loaded, this is most likely because the cross compiler tool chain is not match. Please provide the cross compiler tool chain you used, or provide the compiled environment to us, let us recompile the RIL library.

## 4.2 Rild service in system is not startup

1 Please make sure that the service is added in init.rc file, you can consult the fourth section in Part I to see the concrete.

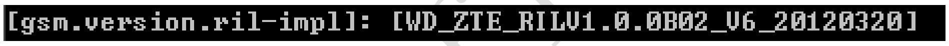
2 Please make sure whether the file of phone.apk is packed into the system by checking if this APK exists in the "/system/app". If the APK is not in system, it will lead to the class phone not created, and the service of rild will not be started too.

## 4.3 Modem port does not exist

Use the command of "ls /dev/ttyUSB*" to check the ports of modem. If there is no ttyUSB prots, it indicates that the ports do not exist. Please make sure whether the modem driver is added in kernel, and the modem PID information is included. To know the concrete, please consult the section of.2.1 and 2.2 in Part I.

## 4.4 How to check the RIL version

Enter the command of getprop in ADB command line. The RIL version will be disaplayed as the following figure shows:



```
[gsm.version.ril-impl]: [WD_ZTE_RILV1.0.0B02_V6_20120320]
```