



Android 系统 RIL 适配用户参考

版本： V0.8

日期： 2013-06-09

法律声明

若接收深圳市中兴物联科技有限公司（以下称为“中兴物联”）的此份文档，即表示您已同意以下条款。若不同意以下条款，请停止使用本文档。

本文档版权所有深圳市中兴物联科技有限公司，保留任何未在本文档中明示授予的权利。文档中涉及中兴物联的专有信息。未经中兴物联事先书面许可，任何单位和个人不得复制、传递、分发、使用和泄漏该文档以及该文档包含的任何图片、表格、数据及其他信息。

ZTE Welink 是中兴物联的注册商标。中兴物联的名称和标志是中兴物联的商标或注册商标，同时中兴物联是中兴通讯的全资子公司，授权使用中兴通讯的注册商标。在本文档中提及的其他产品或公司名称可能是其各自所有者的商标或注册商标。在未经中兴物联或第三方权利人事先书面同意的情况下，阅读本文档并不表示以默示、不可反言或其他方式授予阅读者任何使用本文档中出现的任何标记的权利。

本产品符合有关环境保护和人身安全方面的设计要求，产品的存放、使用和弃置应遵照产品手册、相关合同或相关国法律、法规的要求进行。

本公司保留在不预先通知的情况下，对此手册中描述的产品进行修改和改进的权利；同时保留随时修订或收回本手册的权利。

本用户手册中如有文字不明之处，请您及时向本公司或者代理商、销售商咨询。

修改记录

版本号	拟制/修改日期	问题描述	拟制人	修改内容
0.0	2011.10.17	创建该文档	张毅乐 姜金辉	创建文档
0.1	2011.11.22	语音通道配置功能	张毅乐	添加 3.1 节内容
0.2	2012.01.22	针对 android2.3 新增说明	姜金辉	修改 7.2.3.3
0.3	2012.03.05	新增 CDMA 分支第 8、9 节及 WCDMA 分支第 4 节	姜金辉 张毅乐	CDMA 分支第 8、9 节及 WCDMA 分支第 4 节
0.4	2012.03.06	Framework 层 AT 发送功能	张毅乐	添加 WCDMA 第 5 节，Framework 层 AT 发送功能
0.5	2012.05.16	Android4.0CDMA 分支增加 APN 设置应对彩信	姜金辉	添加 Android4.0CDMA 分支增加 APN 设置，6.4 部分
0.6	2012.05.24	整理文档	张毅乐	1 添加 RIL 驱动集成部分 2 添加常见问题处理（Q&A）部分
0.7	2012.07.17	1 针对三星 smdkv210 平台及 NVIDIA cardhu14r7 平台 Android4.0 版本上 CDMA 分支的新增功能 2 修改并完善文档	罗平波	2.4 节 android4.0 英文级联短信发送功能修改 6.3.3 节 完善修改以前 apn 设置部分 7.2 节 Android4.0CDMA 分支增加呼叫等待与呼叫转移功能 9.1 节 保存 SIM 卡中文联系人修改
0.8	2013.06.09	更新文档模板		更新了文档格式、页眉页脚及修改封面 logo 修改法律声明

目录

修改记录.....	II
目录	IV
第一部分：RIL 驱动集成	1
1 驱动版本说明.....	1
1.1 目录结构.....	1
1.2 文件说明.....	1
2 添加内核模块.....	1
2.1 添加驱动.....	1
2.2 添加设备 VID 和 PID	2
2.3 PPP 组建添加	2
3 集成驱动文件.....	2
3.1 集成拨号脚本.....	2
3.2 修改系统配置.....	3
4 RIL 的配置使用	7
5 调试方法.....	8
第二部分：WCDMA 模块适配系统修改	8
特别说明.....	8
1 信号强度查询.....	8
2 彩信发送和接收.....	9
2.1 需要修改的文件.....	9
2.2 相关的 apn 的设置	10
3 SIM 卡操作	11
3.1 保存 SIM 卡中文联系人.....	11
4 语音业务.....	12
4.1 配置语音通道.....	12
5 数据业务.....	13
5.1 自定义数据拨号的号码.....	13
6 Framework 层发送 AT 命令	14
第三部分：CDMA 模块适配系统修改	17
特别说明.....	17
1 CDMA/GSM 分支切换方法	18
2 短信相关问题.....	19
2.1 短信回执.....	19
2.2 短信特殊字符发送.....	19
2.3 纯英文级联短信的接收.....	21
2.4 使用 UNICODE 编码发送纯英文级联短信.....	22
3 STK 功能.....	24
3.1 适用条件.....	24
3.2 代码下载.....	25
3.3 修改方法.....	25
4 彩信发送/接收	26

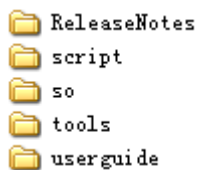
5 信号强度查询.....	30
6 APN 设置修改说明.....	31
6.1 介绍.....	31
6.2 android2.2 及 android2.3 相关文件及修改方法.....	32
6.2.1 xml 文件修改, 增加 APN 相关处理.....	32
6.2.2 java 文件相关修改.....	33
6.3 Android4.0 CDMA 分支增加 APN 设置.....	49
6.3.1 CdmaDataConnection.java 文件相关修改.....	49
6.3.2 CdmaDataConnectionTracker.java 文件相关修改.....	50
6.3.3 RuimRecord.java 文件相关修改.....	61
6.3.4 DataConnectionTracker.java 文件相关修改.....	62
6.3.5 xml 文件相关修改.....	62
6.4 编译、烧写、运行.....	62
7 CDMA 分支增加呼叫等待和呼叫转移的相关修改.....	64
7.1 介绍.....	64
7.2 相关文件及修改.....	66
7.2.1 xml 文件新增.....	66
7.2.2 xml 修改文件.....	66
7.2.3 java 文件新增.....	69
7.2.4 java 文件修改.....	81
7.3 编译、烧写、运行.....	87
8 语音业务.....	88
8.1 配置语音通道.....	88
8.2 通话计时.....	88
9 SIM 卡操作.....	92
9.1 保存 SIM 卡中文联系人.....	92
10 Framework 层发送 AT 命令.....	92
第四部分: 常见故障处理 (Q&A).....	92
特别说明:.....	92
1 数据.....	92
1.1 未设置 APN.....	92
1.2 数据连接未启用.....	92
1.3 init.gprs.pppd 脚本格式不正确。.....	93
1.4 检查拨号脚本的权限.....	93
2 短信.....	93
2.1 短信无法发送和接收, 提示存储空间已满.....	93
3 语音.....	93
3.1 Android4.0.3 上无法进行语音呼叫.....	93
4 其它.....	93
4.1 RIL 库无法加载.....	93
4.2 系统的 rilc 服务未启动.....	94
4.3 模块端口不存在.....	94
4.4 如何查看版本号.....	94

第一部分：RIL 驱动集成

1 驱动版本说明

1.1 目录结构

正式发布的 ANDROID 系统 RIL 驱动的文件结构图如下：



- ReleaseNotes
- script
- so
- tools
- userguide

1.2 文件说明

驱动版本中文件说明：

/ReleaseNotes 代码的修改记录和说明

ReleaseNotes.txt

/script 拨号脚本

init.gprs-pppd

ip-up-ppp0.c

ip-down-ppp0.c

/tools 调试工具

AT 命令发送工具。

BP 的 LOG 抓取工具。

AP 的 LOG 抓取工具。

/userguide 使用说明

Android 系统 RIL 适配用户参考 v0.x.pdf

2 添加内核模块

2.1 添加驱动

模块驱动的添加需要配置系统 android 系统内核，配置方法如下：

cd kernel

make menuconfig

device drivers--->usb support--->usb serial converter support

选中如下组件：

USB driver for GSM and CDMA modems

2.2 添加设备 VID 和 PID

找到内核源码文件 option.c(一般情况下, 路径在..\kernel\drivers\usb\serial\option.c)

在源码中查找如下代码 (蓝色部分), 查找后, 添加红色部分驱动代码:

```
static struct usb_device_id option_ids[] = {
    { USB_DEVICE(0x19d2, 0x1301) },
```

0x19d2 为 ZTE 厂商 ID

0x1301 为 MF206 设备 ID

上面的 ID 根据具体的模块来定, 不同的模块 ID 不同。

2.3 PPP 组建添加

Ril 驱动联网功能底层使用 ppp 协议创建数据链路, 因此需要在内核中配置对 ppp 协议的支持。配置方法如下:

```
cd kernel
```

```
make menuconfig
```

```
device drivers--->network device support--->ppp support
```

选中如下组件:

```
ppp filtering
```

```
ppp support for async serial ports
```

```
ppp support for sync tty ports
```

```
ppp deflate compression
```

```
ppp BSD-compress compression
```

3 集成驱动文件

3.1 集成拨号脚本

1 拷贝文件

拷贝 ip-up-ppp0.c 和 ip-down-ppp0.c 到 ../external/ppp/android.

在 device 目录下创建目录/third_part/zte。

拷贝 init.gprs-pppd 到../device/third_part/zte 目录下。

拷贝 libreference-ril.so 到../device/third_part/zte 目录下。

2 修改文件 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 打包文件到系统中

找到系统编译时候调用的.mk 文件，一般该文件位于/device/xxx/yyy 目录下。其中 xxx 表示平台提供厂商，yyy 表示具体平台型号，比如对于 freescale 的 imx5x 平台，这个目录是 device/fsl/imx5x。在有的代码结构中目录在 vendor/xxx/yyy 下，比如对于 nvidia 的平台该目录为 /vendor/nvidia/harmony。一般该.mk 文件命名方式为“平台名称.mk”或者 AndroidBoard.mk，比如对于飞思卡尔平台的 imx5x 平台该名称为 imx5x.mk，对于 nvidia 的 harmony 平台该文件命名为 AndroidBoard.mk。

在该文件中添加：

```

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/libreference-ril.so

```

注意：当系统编译后，应该在 out 目录下对应平台的 system/etc/下找到 init.gprs-pppd；在 out 目录下对应平台的 system/etc/ppp 下找到 ip-up-ppp0 和 ip-down-ppp0；在 out 目录下对应平台的 system/lib 下找到 libreference-ril.so。如果文件未找到说明相关文件未打包到系统中去，请检查相关配置是否正确。

3.2 修改系统配置

1 修改 init.gprs-pppd 的文件权限，修改代码如下：

路径：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/bluetooth/main.conf" },
    { 00440, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluetooth/input.conf" },
+ { 00777, AID_ROOT, AID_SHELL, "system/etc/init.gprs-pppd" },
    { 00440, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluetooth/audio.conf" },
    /* allow read-write to device-specific configuration files */
    { 00660, AID_BLUETOOTH, AID_BLUETOOTH, "system/etc/bluetooth/*" },
```

2 修改驱动设备文件的权限，修改代码如下：

对于 Android2.2 及以下的版本

路径：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 },
};
```

对于 Android2.3 及以上的版本

修改文件 system/core/rootdir/ueventd.rc,在文件中添加如下内容：

```
/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 修改 system/core/init/property_service.c 文件

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 修改 init.rc 文件

路径：（一般情况下，路径在..\system\core\rootdir\init.rc，具体请见 P10 的“注意”）

```
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
```

```
###添加 rild 服务，-d 表示模块接收 AT 服务的端口参数，-u 表示 DATA 端口。###
```

```
+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
```

```
###添加重启 rild 的机制###
```

```
+on property:ril.reset.rild=1
```

```
+stop ril-daemon
```

```
+start ril-daemon
```

```
+setprop ril.reset.rild 0
```

```
###添加重启模块的机制，其中 ResetModem ()为用户自己实现的重启模块的函数###
```

```
+on property:ril.reset.modem=1
```

```
+/system/bin/ResetModem:
```

```
@@ -329,6 +335,12 @@ service pbap /system/bin/sdptool add --channel=19 PBAP
```

```
disabled
```

```
oneshot
```

```
###添加 pppd_gprs 服务###
```

```
+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 适配升级工具、log 工具

按如下方式修改 init.rc 文件:

```
setprop net.tcp.buffer.size.umts 4094,87380,110208,4096,16384,110208
```

```
setprop net.tcp.buffer.size.edge 4093,26280,35040,4096,16384,35040
```

```
setprop net.tcp.buffer.size.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
```

注意：有些平台厂商维护了自己的 init.rc，因此修改该目录下的 init.rc 可能不会生效。厂商维护的 init.rc 所在路径可能位于 device/xxx/yyy 或者 vendor/xxx/yyy 目录下。其中 xxx 表示平台提供厂商，yyy 表示具体平台型号，如 freescale 的 imx_51bbg 开发平台中，init.rc 位于目录...\device\fsl\imx51_bbg 下。

4 RIL 的配置使用

1 CDMA 模块无卡状态下关闭射频

对于 CDMA 的模块产品，在无卡状态下会不停的根据漫游列表来搜索网络导致模块功耗升高，为了解决这个问题，目前在 RIL 中保留了在无卡状态下关闭射频的开关。当用户在系统中设置属性值 ril.sim.absent.do=close-antenna 后，在 RIL 发现未插卡的情况下会关闭射频。

2 PIN 码 PUK 码剩余次数检测

Android 默认没有给出剩余次数的初始值，即在 PIN 输入的时候，第一次无法得到 PIN 码和 PUK 的剩余次数，目前我们在 RIL 中开机会检查 PIN 码的初始值，并把相关信息写入属性 ril.sim.pin.retry.numbers 和 ril.sim.puk.retry.numbers。上层 APP 可以根据这两个属性值来在 UI 上进行显示。

3 串口+USB 接口设置方法

目前我们的模块支持 USB 接入和串口接入的方式。在默认情况下，使用 USB 接入方式。如果要使用串口+USB 接口方式，需要设置串口的设备路径，通过在系统构建的时候添加属性值 ril.serial.device.path 值为特定串口的方式来实现。

4 CDMA 模块 SIM 卡热拔功能

对于 CDMA 模块，当 SIM 卡热拔除之后会，由于 SIM 卡上的信息已经被保留到了模块中，因此会发现热拔 SIM 卡后，信号强度的显示、网络的注册状态等信息都是好的。有些用户会认为这是一个 BUG，如果要规避这种现象，请在检测到 SIM 卡被拔出的消息后对模块进行复位处理。

5 CDMA 分支 SIM 卡联系人写入功能

在 Android 的默认代码中，SIM 卡联系人写入的时候需要设定存储 ID，该 ID 值表示将联系人信息写入到 SIM 卡的某个存储位置上。目前在 RIL 中提供了一种写入方式，当用户传入的 ID 值为-1 的时候将该联系人写入到 SIM 卡联系人存储区第一个空的位置。

5 调试方法

如果系统调试过程中出现问题，可以通过在 cmd 中输入 adb shell 后进入 ADB 中，并输入以下命令抓取 LOG 来分析：

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

第二部分：WCDMA 模块适配系统修改

特别说明

文中所述方法均在 NVIDIA 平台的 2.2 系统、NVIDIA 平台的 4.0 系统、Freescale 平台的 2.3 系统和三星平台的 4.0 系统上进行了验证。由于目前 Android 系统平台很多，android 版本不断更新，不同平台及版本上的代码有一定的差异性，不能保证文中所述方法在所有平台和版本上都验证通过。同时文中提到的问题，平台厂商也可能已经进行了修正。因此，请根据所使用平台的代码结合该参考文档进行修改。

本文包括 WCDMA 分支和 CDMA 分支两部分，请根据所使用的模块和所需要的功能，自行适配，不需要的补丁不要打，以免引入其它问题。

1 信号强度查询

问题：系统不会主动的查询信号强度（适用于 CDMA/WCDMA）。

分析：由于系统查询信号强度较慢，我们在 RIL 中使用了主动上报，目的是为了让模块起来后马上有信号显示。但是在有些系统中会出现系统不再查询信号强度的问题。其原因是：

- 1 在 SIM 卡 READY 后会调用函数：queueNextSignalStrengthPoll();
- 2 函数 queueNextSignalStrengthPoll() 中判断变量 dontPollSignalStrength 为 false 则继续执行，延时发送消息 EVENT_POLL_SIGNAL_STRENGTH。

在 `CdmaServiceStateTracker.java` 的函数 `handleMessage` 中收到消息 `EVENT_POLL_SIGNAL_STRENGTH` 后会调用函数 `getSignalStrength` 进行信号查询，然后转换为 `REQUEST` 下发下去。

当查询完毕又会有一个事件 `EVENT_GET_SIGNAL_STRENGTH` 上报上来，然后再次调用函数 `queueNextSignalStrengthPoll()` 函数，这样就形成了周期查询。

如果 RIL 主动上报了信号强度，就会有一个消息 `EVENT_SIGNAL_STRENGTH_UPDATE`，那么就会值 `dontPollSignalStrength` 就会为 `true`，这样就在函数 `queueNextSignalStrengthPoll()` 中停止主动的信号查询了。

修改方法：如果要让 RIL 主动上报信号强度后，系统仍然正常查询信号强度，可以修改函数 `queueNextSignalStrengthPoll` 函数，让其不判断 `dontPollSignalStrength` 的值。

修改方法如下：

`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 彩信发送和接收

2.1 需要修改的文件

在 `../frameworks/base/telephony/java/com/android/internal/telephony/gsm/` 目录下的文件 `GsmDataConnection.java` 修改如下：

```
import android.os.SystemClock;
```

```
+import android.os. SystemProperties;
```

```
import android.util.Config;
```

```
.....
```

```
Protected void onConnect(ConnectionParams 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 相关的 apn 的设置

如果要发送彩信，需要设置两个 apn，第一个给访问互联网使用，第二个给彩信使用，需要注意的是给访问互联网使用的 apn 类型要设置为 default，给彩信使用的 apn 类型要设置为 mms，否则无法正常使用彩信功能。

中国联通的彩信参考设置如下表 2.2.1 所示（具体可以拨打 10010 号咨询当地联通运营商）：

表 2.2.1 APN 设置

	上网 APN	彩信 APN
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 SIM 卡操作

3.1 保存 SIM 卡中文联系人

在现有的 android 的电话本存储处理中，当保存联系人到 sim 卡侧，并没有进行中文编码处理，也就是中文联系人是无法保存，或者说保存记录中的中文部分丢失，这样造成了很大的不便。增加新增联系人的中文存储，需要对 framework 层进行一些修改来增加对中文文字的处理，修改文件为：

...\frameworks\base\telephony\java\com\android\internal\telephony\AdnRecord.java。

其中的主要修改集中在函数：

public byte[] buildAdnString(int recordSize) 中，修改内容如下。

@@ -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;
+     }
+ }

```

如果系统已经编译完成，仅需要修改并重新编译 framework 层文件即可修改并查看结果。该修改适用于 android2.2、android2.3 以及 android4.0 的代码适配并已编译调试通过。

4 语音业务

4.1 配置语音通道

问题：目前我们模块提供了几种语音通道的模式，不同的客户可能会根据自己的需要进行配置。

分析：在提供的 RIL 库中留出了这部分的接口，客户需要按照如下内容设置一个属性，目的是告知 RIL 当前使用的是那种语音通道模式。

解决：修改文件../hardware/ril/rild/rild.c，向系统设置属性来告知 RIL 当前使用的语音通道模式。

```

--- 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 数据业务

5.1 自定义数据拨号的号码

问题：一般情况下 WCDMA 的数据拨号号码为*99#，CDMA 为#777。在某些情况下客户要自定义设置拨号号码，这就需要根据本节内容进行修改。

分析：目前在提供的 RIL 库中留出了这部分的接口，客户从上层调用 REQUEST:

RIL_REQUEST_SETUP_DATA_CALL

在这个 REQUEST 中客户将要使用的拨号号码写入到参数 DATA 中，并设置一个属性，这个属性的值表示该参数存在 DATA 数组的哪个参数中。

解决：修改上层调用，将号码传送到 RIL_REQUEST_SETUP_DATA_CALL 的参数 data 中，同时修改 datalen 参数。可参考 ril.h 中的注释代码，如下图所示：

```
/**
 * 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 override.
 * ((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.
```

在 RIL 库中会从传送进来的数据读出自定义号码，使用代码如下：

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

这里 phonenumber_position_init 表示将电话号码存入在 data 所指向数组的哪个元素，它的值需要用户写入系统的属性中，在 RIL 库中会从属性中读取 ril.phonenumber.postion 属性值，从而确定哪个参数表示电话号码。rild.c 文件位于 ../hardware/ril/rild/目录下，这里提供一种实现方法：

```

---    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 @@
    }
}

+ property_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.

```

其中#define NUMBER_POSITION_VENDOR “06”，表示将号码存储在((const char **)data)[6]中，用户可根据需要设定。**注意在宏定义中，请使用两位数来表示。**

6 Framework 层发送 AT 命令

在 RIL 中保留了从 Framework 层下发 AT 命令的接口，用户可以在 Framework 层调用接口函数来实现 AT 命令的发送和响应的接收。其中将 AT 命令分为 singleline、no_result、numeric、multiline 和 multiline_no_prefix 几种。当设置 AT 命令为 singleline 和 multiline 的时候会需要设置参数((char **)data)[2]；如果 AT 命令发送失败会返回错误，否则返回成功；如果设置为 no_result 类型则不会返回 Modem 的响应。RIL 中该部分的实现如下：

```

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\"", 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;
}

```

下面举例说明在收到 sim ready 消息后上层下发 MODEM 语音通话设置命令的方法。在 onSimReady() 函数里面调用函数 sendAtCmd，sendAtCmd 调用系统接口：phone.mCM.invokeOemRilRequestStrings，然后转换为 REQUEST 下发下去，在 handleMessage 中会收到消息 EVENT_GET_OEM_RESPONSE，该消息中包含了对模块 AT 命令的响应。在 ../frameworks/base/telephony/java/com/android/internal/telephony/gsm/ 目录下的文件 SIMRecords.java 代码修改如下：

```

--- 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()

```

第三部分：CDMA 模块适配系统修改

特别说明

文中所述方法均在 NVIDIA 平台的 android2.2 系统、NVIDIA 平台的 android4.0 系统及三星平台的 android4.0 系统上进行了验证。由于目前 Android 系统平台很多，不同平台上的代码有一定的差异性，不能保证文中所述方法在所有平台上都验证通过。同时文中提到的问题，平台厂商也可能已经进行了修正。因此，请根据所使用平台的代码结合该参考文档进行修改。

本文包括 WCDMA 分支和 CDMA 分支两部分，请根据所使用的模块和所需要的功能，自行适配，不需要的补丁不要打，以免引入其它问题。

1 CDMA/GSM 分支切换方法

在文件../frameworks/base/telephony/java/com/android/internal/telephony/RILConstants.java 中定义了第一次启动后默认的电话类型，具体变量为 RILConstants 结构体中的 PREFERRED_NETWORK_MODE。这里建议：如果使用的是 WCDMA 模块，则将该变量初始值设置为 NETWORK_MODE_WCDMA_PREF；若为 CDMA 模块则修该初始值为 NETWORK_MODE_CDMA。单独替换 Framework 层的方法有可能修改不生效，因为启动后的切换可能会导致系统的不稳定，所以修改后应该重新烧写系统镜像并下载。

```
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 */
    .....
}
```

按以上步骤原封不动的修改并且编译下载系统后启动时若出现以下错误：phone 进程意外终止，首先应检查 libreference-ril.so 库是否已经通过 adb push 到/system/lib 下，若是已经 push 仍然出现这个错误可以通过对以下文件进行修改，将 networkMode 写死为 4，再编译整个系统镜像。

修改../frameworks/base/telephony/java/com/android/internal/telephony/PhoneFactory.java 中的 makeDefaultPhone()函数：

```
Log.i(LOG_TAG,"Network Mode set to" + Integer.toString(networkMode));
+ networkMode = 4;
```

```
// Get cdmaSubscription
```

2 短信相关问题

2.1 短信回执

问题：CDMA 设置短信发送报告后，发送成功后，收到的短信回执为“!”。

分析：说明短信状态未被上层 UI 正确解析，请查看以下代码：

```
/frameworks/base/telephony/java/com/android/internal/telephony/cdma/SmsMessage.java
```

若代码中的 `getStatus` 函数如下，则可能是由于在 Framework 层将短信发送状态存储在了上 16 位，而 UI 和 GSM 配套，认为短信存储在下 16 位，因此读不到正确的状态报告。

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

解决方法：修改该函数为：

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

2.2 短信特殊字符发送

问题：发送特殊字符比如‘`’后，其它终端设备接收到为‘?’或者乱码，但是自发自收功能正常（适用于 CDMA，WCDMA 修改方法类似）。

分析：在 Android 上对 CDMA 的编码如果全为字符则采用 7bit-ASCII 编码。目前一些终端设备的处理过程中不能正确处理某些特殊字符的 7bit-ASCII 编码，导致接收异常。另外一种编码格式是 unicode 编码，这种编码格式不存在该问题，但会占用 16 个 bit。我们这里给出的一种做法是如果发现这些特殊字符存在，则使用 unicode 编码。

解决方法：

```
--- 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) {
                            outputStream.write(7, UserData.UNENCODABLE_7_BIT_CHAR);

```

注意：在测试过程中，目前只发现字符 ‘^’ 无法发送。若发现其它字符也出现类似现象，可以在代码中按照该方法做相应修改。

2.3 纯英文级联短信的接收

问题：接收纯英文级联短信后解析错误。

分析：CDMA 接收到的纯英文级联短信一般为 7bit 的 ASCII 类型。相对于单条短信多出了对短信头的解析，同时还要跳过相应的填充位。

解决：修改文件 ../frameworks/base/telephony/java/com/android/internal/telephony/cdma/sms/BearerData.java 中的函数 decode7bitAscii()，具体如下所示：

```
--- 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
 {
     - 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 " + wantedBits +
             " + 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 使用 UNICODE 编码发送纯英文级联短信

问题：当前板侧不支持 7bit-ASCII 编码的纯英文级联短信的发送。

分析：目前使用 UNICODE 编码来暂时解决该问题。

解决：修改如下代码

计算使用 UNICODE 编码时候每条短信的长度信息和消息分割的个数。

--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/BearerData.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();

```

在 android2.2 及 android2.3 中，发现是长短信的时候使用 UNICODE 编码，按如下方式修改文件 CdmaSMSDispatcher.java 中的 sendMultipartText()函数：

--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

```

在 android4.0 中，发现是长短信的时候同样使用 UNICODE 编码，由于 android4.0 的源代码与 android2.2 及 android2.3 有一定的出入，按如下方式修改 SMSDispathcer.java、CdmaSMSDispathcer.java、GsmSMSDispathcer.java 这三个文档：

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

在函数 sendMultiPartText()中，修改如下：

```

-        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);

```

修改文档：

---init/frameworks/base/telephony/java/com/android/internal/telephony/cdma/CdmaSMSDispathcer.java

在该文档中 sendNewSubmitPdu()函数实现后新增函数：

```

+ 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 statusReportRequested bit only for the
+      * last message fragment, this will result in only one
+      * callback to the sender when that last fragment delivery
+      * has been acknowledged. */
+     SmsMessage.SubmitPdu submitPdu = SmsMessage.getSubmitPdu(destinationAddress,
+         uData, (deliveryIntent != null) && lastPart);
+
+     sendSubmitPdu(submitPdu, sendIntent, deliveryIntent);

```

修改文档:

---init/frameworks/base/telephony/java/com/android/internal/telephony/gsm/GsmSMSDispatcher.java

在该文档中 sendNewSubmitPdu()函数实现后新增函数:

```

+ protected abstract void sendMultiSubmitPdu(String destinationAddress, String scAddress,
+     String message, SmsHeader smsHeader, int encoding,
+     PendingIntent sendIntent, PendingIntent deliveryIntent, Boolean lastPart, int msgCount) {
+     SmsMessage.SubmitPdu pdu = SmsMessage.getSubmitPdu(scAddress, destinationAddress,
+         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 STK 功能

3.1 适用条件

在有些平台的 Android2.2 和 Android2.3 的源码中 CDMA Phone 中没有对 STK 部分的相关处理,同时 GSM Phone 中对 STK 的处理是在 package com.android.internal.telephony.gsm.stk 中完成。由于在 CDMA Phone 类的方法和成员变量类型和 GSM Phone 类中的不同,因此不

能直接使用 GSM 的 STK 服务。如果是这种情况则需要对 Framework 层进行修改, 开启 STK 的服务, 并和上层 UI 配合进行工作。

另一种在 Framework 层对 STK 进行处理的服务是 Catservice, 它可以同时使用于 CDMA 和 GSM, 本文介绍如何将该 Catservice 移植到 CDMA 分支上。

在 android4.0 版本中, 源代码中已经包含了 STK 部分的相关处理。不需要再自行修改。

3.2 代码下载

Framework 层的 Cat 服务代码可以在网址:

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

对应的使用 Cat 服务的 STK 应用程序下载地址:

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

3.3 修改方法

将 2 中下载的文件夹 Cat 放在:

/frameworks/base/telephony/java/com/android/internal/telephony/目录下

将 2 中下载的文件夹 Stk 放在:

/android2.2/packages/apps/目录下

修改文件:

Frameworks/base/telephony/java/com/android/internal/telephony/CDMAPhone.java

1 添加: (57) import com.android.internal.telephony.cat.CatService;

2 添加: 增加类 CDMAPhone 的成员变量:

(112) CatService mCcatService

3 添加: 在类 CDMAPhone 的构造函数中增加:

(167) mCcatService = CatService.getInstance(mCM, mRuimRecords, mContext,
mIccFileHandler, mRuimCard);

修改文件:

/frameworks/base/telephony/java/com/android/internal/telephony/baseCommands.java

1 修改: (398) setOnStkSessionEnd->setOnCatSessionEnd

2 修改: (402) unSetOnStkSessionEnd->unSetOnCatSessionEnd

3 修改: (406) setOnStkProactiveCmd->setOnCatProactiveCmd

4 修改: (410) unSetOnStkProactiveCmd->unSetOnCatProactiveCmd

5 修改: (414) setOnStkEvent->setOnCatEvent

6 修改: (418) unSetOnStkEvent->unSetOnCatEvent

7 修改: (422) setOnStkCallSetUp->setOnCatCallSetUp

8 修改: (426) unSetOnStkCallSetUp->unSetOnCatCallSetUp

修改文件:

Framework/base/telephony/java/internal/telephony/CommandsInterface.java

1 修改: (384) setOnStkSessionEnd->setOnCatSessionEnd

2 修改: (385) unSetOnStkSessionEnd->unSetOnCatSessionEnd

3 修改: (396) setOnStkProactiveCmd->setOnCatProactiveCmd

4 修改: (397) unSetOnStkProactiveCmd->unSetOnCatProactiveCmd

- 5 修改: (407) setOnStkEvent->setOnCatEvent
- 6 修改: (408) unSetOnStkEvent->unSetOnCatEvent
- 7 修改: (418) setOnStkCallSetUp->setOnCatCallSetUp
- 8 修改: (419) unSetOnStkCallSetUp->unSetOnCatCallSetUp

修改文件:

/framework/base/telephony/java/android/internal/telephony/gsm/stk/stkService.java

- 1 修改: (163) mCmdIf.setOnStkSessionEnd(this, MSG_ID_SESSION_END, null); ->
mCmdIf.setOnCatSessionEnd(this, MSG_ID_SESSION_END, null);)
- 2 修改: (164) mCmdIf.setOnStkProactiveCmd(this, MSG_ID_PROACTIVE_COMMAND, null); ->
mCmdIf.setOnCatProactiveCmd(this, MSG_ID_PROACTIVE_COMMAND, null);
- 3 修改: (166) mCmdIf.setOnStkEvent(this, MSG_ID_EVENT_NOTIFY, null);->
mCmdIf.setOnCatEvent(this, MSG_ID_EVENT_NOTIFY, null);
- 4 修改: (167) mCmdIf.setOnStkCallSetUp(this, MSG_ID_CALL_SETUP, null);->
mCmdIf.setOnCatCallSetUp(this, MSG_ID_CALL_SETUP, null);
- 5 修改: (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 彩信发送/接收

问题: 彩信无法发送。

分析: 彩信的发送基本过程为: 用户点击发送按钮后, 系统检测到是一条彩信, 会先断开当前类型为 default 的 APN 数据连接, 然后连接 mms 类型的 APN, 将短信 PUSH 到网络上后, 再恢复 default 数据连接类型。

解决: CDMA 分支缺少 APN 设置部分, 请参考本文档“APN 设置修改说明”一节来添加这一部分。添加完成后应该可以发送彩信了, 注意要设置彩信的 APN, APN 类型为 MMS, 同时数据网络连接时 APN type 的类型为 default, 如表 4.1 所示为中国电信的一种 APN 设置方法。(具体可以拨打 10000 号咨询当地电信运营商):

表 4.1 APN 设置

	上网 APN	彩信 APN
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

在文件：Frameworks/base/telephony/java/com/android/internal/telephony/phone.java 中定义了 APN 的类型，具体可查看源代码。

默认的代码中在接收中国电信彩信过程中会出现问题，主要是电信的封装类型和 Android 源码上层解析不一致，电信增加了一层封装。修改方法如下：

1 在文件 telephony/java/com/android/internal/telephony/cdma/sms/BearerData.java 的函数：decodeUserData (BearerData bData, BitwiseInputStream inStream)后面添加如下函数，其中增加了对于中国电信彩信通知的特殊处理，跳过了一层封装：

```
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 在文件 telephony/java/com/android/internal/telephony/cdma/sms/BearerData.java 的函数:

public static BearerData decode(byte[] smsData)后面添加如下函数，这个函数调用了上面的添加的 decodeUserData:

```

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 添加中国电信的 teleServiceid

```

---
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 修改如下文件，在解析中国电信彩信的时候调用上面定义的函数。

```

---    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 根据 teleServiceId 判断是彩信，并报告给上层处理。

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.java

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 信号强度查询

问题：系统不会主动的查询信号强度。

分析：由于系统查询信号强度较慢，我们在 RIL 中使用了主动上报，目的是为了让模块起来后马上有信号显示。但是在有些系统中会出现系统不再查询信号强度的问题。其原因是：

- 1 在 SIM 卡 READY 后会调用函数：queueNextSignalStrengthPoll();
- 2 函数 queueNextSignalStrengthPoll() 中判断变量 dontPollSignalStrength 为 false 则继续执行，延时发送消息 EVENT_POLL_SIGNAL_STRENGTH。

在 CdmaServiceStateTracker.java 的函数 handleMessage 中收到消息 EVENT_POLL_SIGNAL_STRENGTH 后会调用函数 getSignalStrength 进行信号查询，然后转换为 REQUEST 下发下去。

当查询完毕又会有一个事件 EVENT_GET_SIGNAL_STRENGTH 上报上来，然后再次调用函数 queueNextSignalStrengthPoll() 函数，这样就形成了周期查询。

如果 RIL 主动上报了信号强度，就会有一个消息 EVENT_SIGNAL_STRENGTH_UPDATE，那么就会值 dontPollSignalStrength 就会为 true，这样就在函数 queueNextSignalStrengthPoll() 中停止主动的信号查询了。

修改方法：

```

Frameworks/base/telephony/Java/com/android/internal/telephony/cdma/CdmaServiceStateTracker.java
private void

```

```
queueNextSignalStrengthPoll() {  
    - if (dontPollSignalStrength || (cm.getRadioState().isGsm())) {  
    +if (cm.getRadioState().isGsm()) {  
        // The radio is telling us about signal strength changes  
        // we don't have to ask it  
        return;  
    }  
    Message msg;  
    msg = obtainMessage();  
    msg.what = EVENT_POLL_SIGNAL_STRENGTH;  
    // TODO Don't poll signal strength if screen is off  
    sendMessageDelayed(msg, POLL_PERIOD_MILLIS);  
}
```

6 APN 设置修改说明

6.1 介绍

在有些平台的 Android2.2 和 Android2.3 的源码中 CDMA Phone，在普通的 android 代码 CDMA 分支中，没有对 APN 设置部分，即编译出的系统在菜单 settings->Wireless&networks->Mobile networks 里面的列表中，不会显示如图 6.1.1 所示红色部分内容。因此也不存在点击“Access Point Names”后的界面及 menu 项，新增这个功能需要在 app 层级 framework 层增加部分代码。

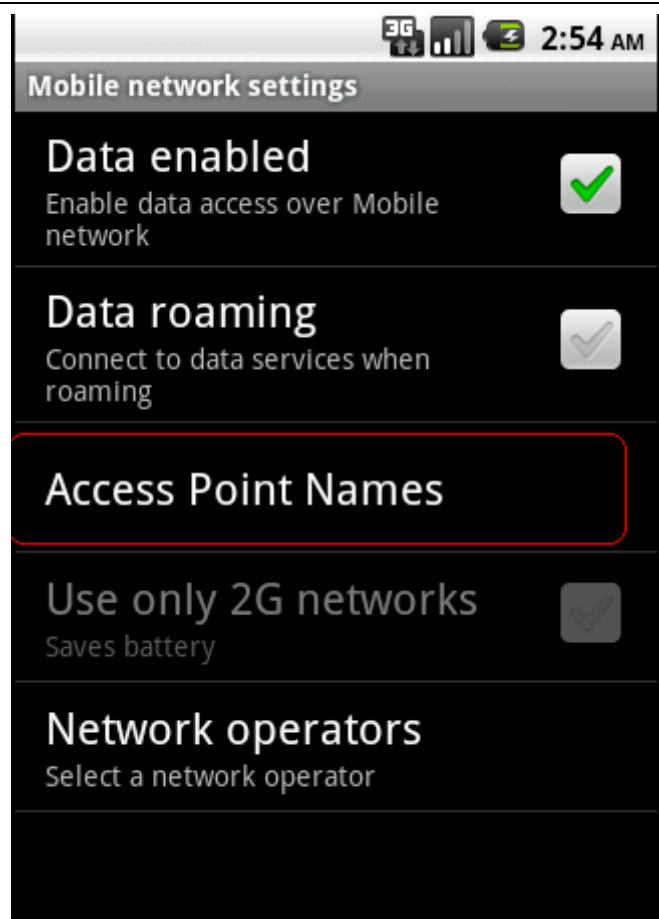


图 6.1.1 APN 功能入口界面

6.2 android2.2 及 android2.3 相关文件及修改方法

主要涉及如下两个部分文件：

APP 层显示相关部分文件

packages/apps/Phone/res/xml/cdma_options.xml

内部处理部分相关文件。

frameworks/base/telephony/java/com/android/internal/telephony/cdma 中的
CdmaDataConnection.java

CdmaDataConnectionTracker.java

RuimRecords.java

另外还有

frameworks/base/telephony/java/com/android/internal/telephony/gsm 中的
ApnSetting.java

6.2.1 xml 文件修改，增加 APN 相关处理

6.2.1.1 cdma_options.xml 文件

cdma_options.xml 文件，增加 APN 相关处理在文件中

xmlns:settings="http://schemas.android.com/apk/res/com.android.phone">这句话之后增加
如下内容。

```

<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>

```

在增加了如上部分代码后，就会如图 6.1.1 一样，显示红色圈住部分的内容，进入后即可进行 APN 设置。

6.2.2 java 文件相关修改



6.2.2.1 CdmaDataConnection.java 文件修改

public class CdmaDataConnection extends DataConnection 类中的
protected void onConnect(ConnectionParams cp)函数中，关于设置

```
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);
```

修改为：

```
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), (在 android2.3 代码
中需要增加 RILConstants.SETUP_DATA_PROTOCOL_IP,) msg);
```

这样调用 RIL 的 setupDataCall 进行网络连接的时候，使用的就不再是 NULL 的空内容了而是我们通过 apn 设置所设置的内容了。

6.2.2.2 CdmaDataConnectionTracker.java 文件修改

其中修改的内容较多，新增的函数有

```
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)
还有其他一些函数也进行了修改内容如下
新增 ApnChangeObserver 及其它变量
//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);

```

在函数 CdmaDataConnectionTracker(CDMAPhone p)中
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.

```

在函数 public void dispose()中

```

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() {

```

在函数 `getActiveApnTypes` 中

```

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;

```

在 `protected String getActiveApnString()` 函数中

```

+protected String getActiveApnString() {
+ String result = null;
+ if (mActiveApn != null) {
+ result = mActiveApn.apn;
+ }
+ return result;
+}
+
/**
* The data connection is expected to be setup while device

```

在函数 `private boolean trySetupData(String reason)` 中

```

&& (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_UNKNOWN_APN);
+         return false;
+     } else {
+         log ("Create from allApns : " + apnListToString(allApns));
+     }
+}

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

```

```
return setupData(reason);
```

```
} else {
```

在函数 private void cleanUpConnection 中
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;
    }
}

```

在函数 private boolean setupData(String reason)中

```

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);

```

在函数 onRecordsLoaded 中

```

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

```

在函数 onEnableNewApn 中

```

        protected void onEnableNewApn() {
+            mRetryMgr.resetRetryCount();
+            Log.d(LOG_TAG, "onEnableNewApn ");
            cleanUpConnection(true, Phone.REASON_APN_SWITCHED);
        }

```

在函数 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);
+                }
+            } else {
+                SystemProperties.set("gsm.defaultpdcontext.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(apnTypeId(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.
```

在函数 protected void onDisconnectDone(AsyncResult ar)中增加

```
    phone.notifyDataConnection(reason);
```

```
+    mIsApnActive = false;
```

```
    mActiveApn = null;
```

```
    if (retryAfterDisconnected(reason)) {
```

在函数 public void handleMessage (Message msg)中增加处理

```
        onRestartRadio();
```

```
        break;
```

```
+ case EVENT_APN_CHANGED:
```

```
+     onApnChanged();
```

```
+     break;
```

```
        default:
```

以下均为新增函数处理：

```
+/**
```

```
+ * 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_UNKNOWN_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 (!canSetPreferredApn) {
+        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;
+         } else {
+             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;
+}

```



```
+ 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();
+       }
+   }
+}

```

```

+    }
+ }
+}
+
+ }

```

此文件中处理，主要增加了对于 apn 列表的建立，及 APN 变化时的处理。

着重的修改处理在于 setupData 函数里面的修改，其他处理都是围绕这个核心在修改。

6.2.2.3 ApnSetting.java 文件修改

frameworks\base\telephony\java\com\android\internal\telephony\gsm\ ApnSetting.java 文件中

由于 cdma 分支需要用到 apnsetting.java 中的内容，所以，将此文件中原有的普通变量，变 public 变量，以便使用。

如

String carrier;	->	public String carrier;
String apn;	->	public String apn;
String proxy;	->	public String proxy;
String port;	->	public String port;
String mmesc;	->	public String mmesc;
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 分支增加 APN 设置

6.3.1 CdmaDataConnection.java 文件相关修改

在 CdmaDataConnection.java 中的 CdmaDataConnection 类中 新增内容如下。

修改内容如下 在 onConnect 方法中将

phone.mCM.setupDataCall 函数中的三个参数 NULL 分别改为

```

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 CdmaDataConnectionTracker.java 文件相关修改

新增代码如下

```
boolean failNextConnect = false;
/*ZTE_JJH_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_JJH_CDMAAPN_001 end*/
```

CdmaDataConnectionTracker 函数中 增加如下代码

```
        createAllDataConnectionList();
/*ZTE_ZHAOYI_CDMAAPN_001 begin*/
        apnObserver = new ApnChangeObserver();
        p.getContext().getContentResolver().registerContentObserver(
            Telephony.Carriers.CONTENT_URI, true, apnObserver);
/*ZTE_ZHAOYI_CDMAAPN_001 end*/
dispose 函数中 增加如下代码
mPhone.getContext().getContentResolver().unregisterContentObserver(this.apnObserver);
/*ZTE_JJH_CDMAAPN_001*/
        destroyAllDataConnectionList();
```

在 setState 函数中修改内容为如下。

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

```
}
```

在 setState 函数后面增加函数如下

```
/*ZTE_JJH_CDMAAPN_001 begin*/
    @Override
    protected boolean isApnTypeActive(String type) {
        return mActiveApn != null && mActiveApn.canHandleType(type);
    }
/*ZTE_JJH_CDMAAPN_001 end*/
```

isApnTypeAvailable 函数修改为如下内容

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

在 trySetupData 函数中新增内容如下

```
if ((mState == State.IDLE || mState == State.SCANNING) &&
    isDataAllowed() && getAnyDataEnabled() && !isEmergency()) {
/*ZTE_JJH_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;

```

在函数 cleanUpConnection 中

```

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

```

在函数 setupData 中

```

/*ZTE_JJH_CDMAAPN_001 begin*/
log("CDMA APN setupData");
    ApnSetting apn= getNextApn();
    if (apn == null) return false;
    log("CDMA APN setupData apn != null");
/*ZTE_JJH_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*/

```

修改代码如下:

```

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

```

在函数 onRecordsLoaded 中新增代码如下:

```

createAllApnList(); /*ZTE_JJH_CDMAAPN_001*/
if (mState == State.FAILED) {

```

修改函数 onDataSetupComplete 为:

```

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
    }

```



```

    * 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))),  
        );  
            result.add(apn);  
        } while (cursor.moveToNext());  
    }  
    return result;  
}
```

```

        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 (!canSetPreferredApn) {
        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 buildWaitingApns");
        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 RuimRecord.java 文件相关修改

修改 onAllRecordsLoaded()函数:

```

Protected 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);
}

```

新增函数 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 DataConnectionTracker.java 文件相关修改

修改 onEnableApn()函数

```
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 xml 文件相关修改

xml 文件修改同 android 其他版本修改。

6.4 编译、烧写、运行

编译系统，确保 app 层和 Framework 层修改的代码都编译进系统中。下载系统镜像，加载 CDMA 分支的 RIL 库文件，使用 cdma 模块及 sim 卡就可以在 settings 中找到相应的内容了。已有 apn 信息可显示位如图 6.4.1 所示：

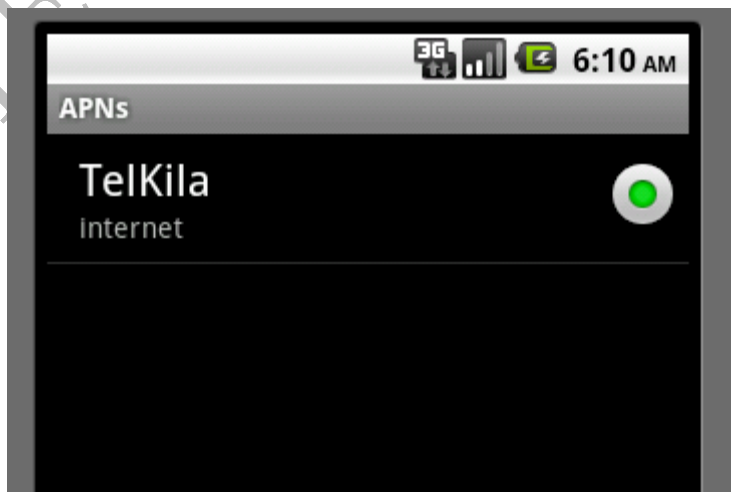


图 6.4.1 APN 显示界面

可以新增 apn，新增 apn 界面如下图所示：

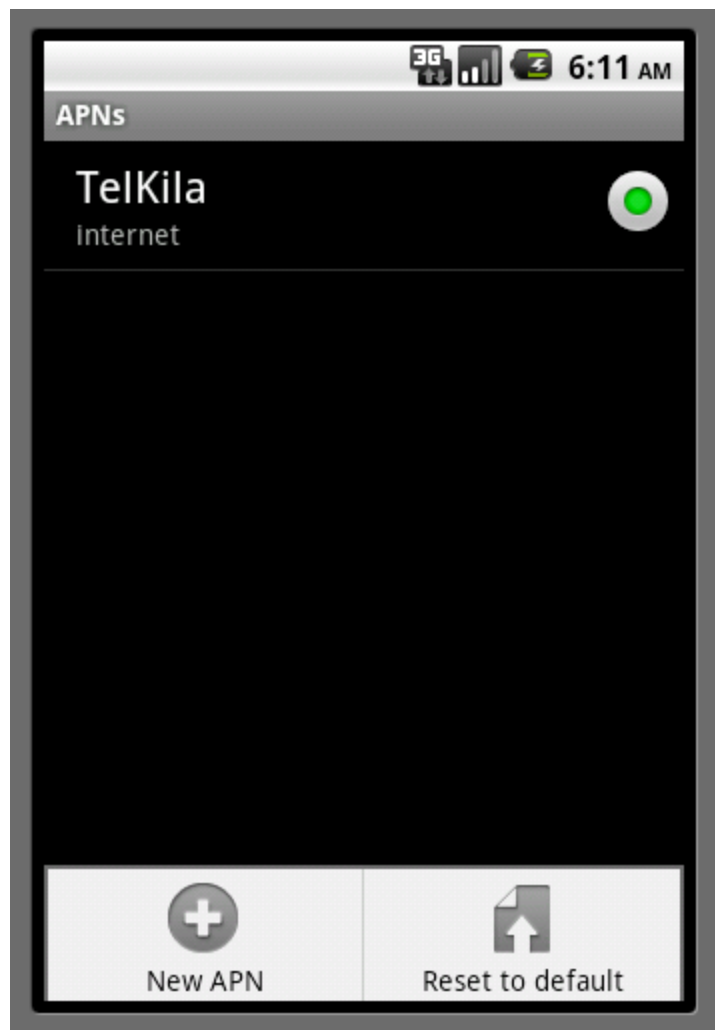


图 6.4.2 新增 APN 显示界面

新增的 apn 可以保存在列表中。并可以选择和编辑，界面如图 6.4.3 所示。

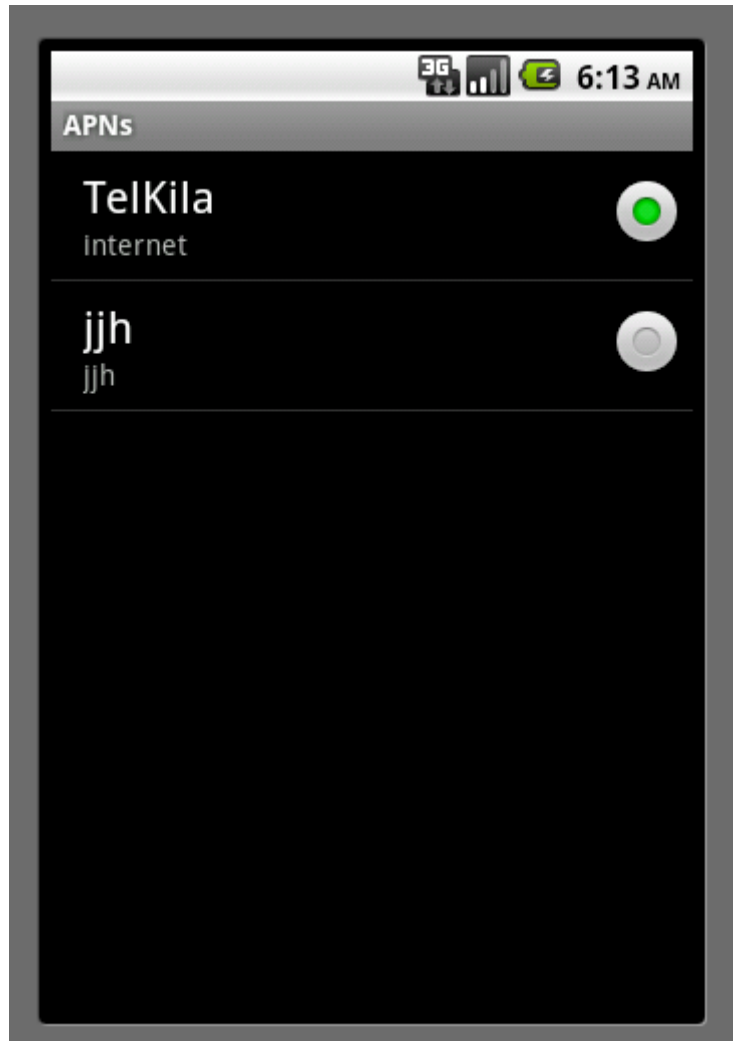


图 6.4.2 选择和编辑 APN 界面

7 CDMA 分支增加呼叫等待和呼叫转移的相关修改

7.1 介绍

在有些平台（如三星、NVIDIA、freescall 等）的 Android2.2 及 android4.0 的 CDMA 分支的源码中，没有对 CDMA 的呼叫等待和呼叫转移的相关处理。

在 android2.2 及 android2.3 源码中，CDMA 分支的菜单 settings->Call settings 列表里，是不会显示如图 7.1.1 所示“呼叫转移”、“呼叫等待”部分的内容。因此图 7.1.2 所示的呼叫等待页面的内容更是不会显示，要想实现呼叫等待与呼叫转移的功能，不仅需要在 APP 层增加一些处理，framework 层的处理也是需要相应增加的。

而在 android4.0 系列源码中，“通话设置”界面的入口位置与 android2.x 系列区别较大，在进入拨号界面后，点击 menu 键，出现“通话设置”界面，与 android2.x 系列相似的是同样不具备“呼叫转移”、“呼叫等待”功能，本节主要介绍在 android2.2、android2.3 以及 android4.0 上实现该功能所需增加的代码。



图 7.1.1 通话设置页面

其中呼叫转移选项中列表如下：



图 7.1.2 呼叫转移设置页面

由于 CDMA 呼叫转移和呼叫等待功能是使用拨号的方式来发起，状态记录在网络侧。

拨号的号码由转移类型（等待）码和转移呼叫号码构成。转移类型（等待）码由各个运营商定义，同一种类型的呼叫转移开启或禁用码也是不同的，因此一个转移类型码唯一标示了某种类型的呼叫转移的开启或者关闭。转移呼叫号码是有由用户自己编辑的，该号码是在满足该项呼叫转移时进行转移的号码。而来电等待并不需要用户自己编辑电话号码，只需要用两个等待码来标示是开启呼叫等待还是关闭呼叫等待。所以只是需要在 app 层进行相关修改，而无需对 framework 层进行过多修改，仅需增加一个属性。下面介绍一下相关修改，app 层即修改 `\packages\apps\Phone` 中内容，framework 层则修改 `\frameworks\base\telephony\java\com\android\internal\telephony\CommandsInterface.java` 文件即可，其他部分也有相关文件需要修改。

7.2 相关文件及修改

7.2.1 xml 文件新增

7.2.1.1 新增文件 `cdma_callforward_options.xml`

在 `\packages\apps\Phone\res\xml\` 中加入新的文件来进行 cdma 呼叫转移的选项页面，新增文件为 `cdma_callforward_options.xml`，此文件大部分内容与 `callforward_options.xml` 基本相同，但最后部分增加了关于取消呼叫转移部分的处理，新增页面内容如下（[Android2.2、android2.3 以及 android4.0 处理相同](#)）：

```

    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 xml 修改文件

7.2.2.1 修改文件 `cdma_call_options.xml`

在 `..\packages\apps\Phone\res\xml\cdma_call_options.xml`，此文件中原有内容仅为 Voice Privacy，但由于需要增加呼叫等待和呼叫转移的处理所以将其修改为如下（[仅适用于](#)

Android2.2 以及 android2.3 系列源代码修改):

```

        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 修改文件 cdma_call_privacy.xml

该修改仅适用于 [android4.0 源代码修改](#)，因为在 android2.x 和 android4.0 源代码中，cdma_call_privacy.xml 与 cdma_call_options.xml 两个文件的内容及其功能均相同，仅文件名不同，按如下方式修改文件：..\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 修改文件 pref_dialog_editphonenum.xml

在 \ packages\apps\Phone\res\layout\pref_dialog_editphonenum.xml 中加入新的 cdma 的呼叫转移和呼叫等待的对话框页面。其内容如下 ([Android2.2](#)、[android2.3](#) 以及 [android4.0](#) 处理相同):

```

    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 修改文件 attrs.xml

在 \ packages\apps\Phone\res\values\attrs.xml, 此文件中在原有内容基础上增加了取消全部呼叫转移的属性。修改如下 ([Android2.2](#)、[android2.3](#) 以及 [android4.0](#) 处理相同):

```
<enum name="not_reachable" value="3" />
```

```

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

```

7.2.2.5 修改文件 strings.xml

在 \ packages\apps\Phone\res\values\strings.xml，此文件中增加新增的呼叫等待，呼叫转移等相关文字（[Android2.2](#)、[android2.3](#) 以及 [android4.0](#) 处理相同）：

```

+   <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 修改文件 AndroidManifest.xml

在 \ packages\apps\Phone\AndroidManifest.xml 在此文件中增加如下处理，增加呼叫转移的显示（[android2.2](#)、[android2.3](#) 以及 [android4.0](#) 处理相同）：

```

    <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 java 文件新增

下面介绍 java 的文件修改。



7.2.3.1 新增文件 CdmaCallForwardOptions.java

在\packages\apps\Phone\src\com\android\phone 中，增加呼叫转移相关处理的两个文件 CdmaCallForwardOptions.java 与已有的文件 GsmUmtsCallForwardOptions.java 有较为相似，其不同为如下（android2.2、android2.3 以及 android4.0 处理除了以下蓝色部分其余均相同）：

```
+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);//适用于 android2.x
+  addPreferencesFromResource(R.xml.cdma_callforward_privacy);//适用于 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 新增文件 CanelCallForwardPreference.java

新增文件 CanelCallForwardPreference.java 与原有文件 CallForwardEditPreference.java 较为相似，其修改内容如下（[android2.2](#)、[android2.3](#) 以及 [android4.0](#) 处理相同）：

```

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;
+ }

```

在函数 `protected void onDialogClosed(boolean positiveResult)` 中：

```

    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_JJH_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_JJH_CDMAAPN_001 end*/
+      this.mButtonClicked = DialogInterface.BUTTON2;
+  }

```

在函数 `handleCallForwardResult` 中，修改如下：

```

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

```

在函数 `updateSummaryText()` 中，修改为如下所示：

```

+/**
+    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;

```

在函数 private void handleGetCFResponse(Message msg)中

```

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();

```

在 handleSetCFResponse 函数中

```

    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 新增文件 CdmaCallWaiting.java

(在 [adnroid2.3](#)、[android4.0](#) 代码中和部分没有此文件的 [android2.2](#) 版本中添加此文件)，路径为 `packages\apps\Phone\src\com\android\phone\`，文件内容如下：

```

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 java 文件修改

7.2.4.1 修改文件 CallForwardEditPreference.java

\packages\apps\Phone\src\com\android\phone\CallForwardEditPreference.java 文件修改内容如下 ([android2.2](#)、[android2.3](#) 与 [android4.0](#) 修改均相同):

```

Phone phone;
+ int mode;
+ Context mContext;

```

CallForwardEditPreference 构造函数中:

```

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

```

在函数 protected void onDialogClosed(boolean positiveResult)中:

```

CommandsInterface.CF_ACTION_REGISTRATION :
CommandsInterface.CF_ACTION_DISABLE;

+
+boolean choice = isToggled() || (mButtonClicked == DialogInterface.BUTTON_POSITIVE);
+ if(choice == true){

```


新增函数

```
+ 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:  
+                 break;  
+         }  
+     }  
+ }
```

```

+         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 修改文件 EditPhoneNumberPreference.java

\packages\apps\Phone\src\com\android\phone\EditPhoneNumberPreference.java 文件中的修改内容如下 ([android2.2](#)、[android2.3](#) 与 [android4.0](#) 修改相同):

```

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);

```

在函数 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;
+         }
    }

```

新增函数：

```

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

```

在函数 `protected void onBindDialogView(View view)` 中：

```

    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");
+             }
+         }
+     });
+ }
+ }
+ }

```

在函数 onPrepareDialogBuilder(AlertDialog.Builder builder)中，修改如下：

```

// 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);
+     }
+ }

```

最后在 framework 层进行的修改如下仅需在

frameworks\base\telephony\java\com\android\internal\telephony\CommandsInterface.java 文件

中修改即可（[android2.2](#)、[android2.3](#) 与 [android4.0](#) 修改相同）：

```
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 编译、烧写、运行

编译系统，确保 app 和 Framework 层修改的代码都编译进系统中。下载系统镜像，加载 CDMA 分支的 RIL 库文件，使用 cdma 模块及 sim 卡就可以在 settings 中找到相应内容了。

在点击了呼叫转移后会弹出如下 7.3.1 所示对话框。



图 7.3.1 呼叫转移界面

取消呼叫转移时如图 7.3.2 所示：



图 7.3.2 取消呼叫转移界面

呼叫等待时弹出对话框如 7.3.3 所示：



图 7.3.3 呼叫等待界面

8 语音业务

8.1 配置语音通道

问题：目前我们模块提供了几种语音通道的模式，不同的客户可能会根据自己的需要进行配置。

分析：在提供的 RIL 库中留出了这部分的接口，客户需要按照如下内容设置一个属性，目的是告知 RIL 当前使用的是那种语音通道模式。

解决：修改文件/hardware/ril/rild/ril.c，向系统设置属性来告知 RIL 当前使用的语音通道模式。

```
--- 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 通话计时

中国电信规范要求终端有两种通话计时：呼叫时常和通话时常。呼叫时长从对方振铃状

态开始计时，通话时长从对方摘机状态开始。

在目前的电信 CDMA 网络下，有些地方的摘机信号从网络侧无法拿到，因此目前我们给出的方案是：在开始呼叫、对方振铃的时候开始计时，这个时间是呼叫时常，如果网络侧会下发对方摘机信号，则清空计时时间，开始通话计时。如果没有拿到摘机信号，则一直显示呼叫时常。这需要 Framework 层进行同步的修改，代码如下：

1 添加一个事件来标识对方摘机事件，具体代码如下：

```
--- 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() {
```

2 注册 EVENT_CDMA_LINE_CONTROL_INFO 事件，并进行处理，当收到该消息后进行通话计时的清空。

```
--- 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 在 `cdmaConnetction.java` 文件中添加清空计时函数:

```

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 SIM 卡操作

9.1 保存 SIM 卡中文联系人

参见 [WCDMA 分支第 3 节](#)

10 Framework 层发送 AT 命令

参见 WCDMA 第 6 节内容。

第四部分：常见故障处理（Q&A）

特别说明：

如果您之前适配过其它厂家的模块产品，建议先删除之前适配过程中修改的文件，再进行 ZTE 模块适配工作。

1 数据

1.1 未设置 APN

APN 的设置：在【设置-无线网络和设置-移动网络-接入点名称】。请咨询相关运营商或者参考手机进行设置。

1.2 数据连接未启用

在【设置-无线网络和设置-移动网络-启用数据连接】中查看数据连接是否已经启用，若数据连接未启用，则数据连接不会进行。

1.3 init.gprs-pppd 脚本格式不正确。

所提供的 init.gprs-pppd 脚本为 UNIX 格式，若使用 Windows 编辑器打开后有可能系统会将其转换为 DOS 格式，当该文件被转换为 DOS 格式后，这个脚本在 Android 系统上就运行不起来了。因此一定要确保该文件为 UNIX 格式。

对于 UNIX 格式的文件，在使用 UltraEdit-32 打开的时候会提示是否将其转换为 DOS 格式。同时使用 UltraEdit-32 的【文件-格式转换】可以将 DOS 格式转换为 UNIX 格式。

1.4 检查拨号脚本的权限

查看 ip-up-ppp0 和 ip-down-ppp0 的权限。这两个文件应该位于系统的/system/etc/ppp 目录下，同时具有可执行的权限，请确认。

查看 init.gprs-pppd 的权限。这两个文件应该位于系统的/system/etc 目录下，同时具有可执行的权限，请确认。

2 短信

2.1 短信无法发送和接收，提示存储空间已满

请确认/data 目录是否可以读写，同时写入到/data 目录中修改的内容是否可以在断电重启后保留。

3 语音

3.1 Android4.0.3 上无法进行语音呼叫

这是某些硬件平台在Android4.0.3上的一个BUG，请咨询平台提供厂商解决。

语音功能的验证可以在系统生成的时候通过设置属性ro.sf.lcd_density=160或者大于160后进行电话拨打，验证功能。由于这样会使得屏幕分辨率变大，因此非根本解决方案。

4 其它

4.1 RIL 库无法加载

如果发现我们提供的 RIL 库文件无法被系统调用起来（即无任何 ZTE 的 LOG 打印），但是使用系统默认的 RIL 库文件可以正常加载，则很有可能是交叉编译链不对，请提供您所使用的交叉编译链或者编译环境给我们重新编译库文件。

4.2 系统的 ril 服务未启动

1 请确保 init.rc 中已经添加服务，详细参考第一部分，第 4 节中相关内容。

2 检查 phone.apk 是否打包进系统，在/system/app 下查看是否有该 APK。如果该 APK 未打包进系统，会导致 phone 类不会被创建，ril 也不会被启动。

4.3 模块端口不存在

使用 ls /dev/ttyUSB* 查看模块端口，发现无 ttyUSB 端口，说明端口不存在。请确认是否在内核中添加了模块驱动，并包含了模块的 PID 信息。具体详见 2.1 和 2.2 节内容。

4.4 如何查看版本号

在 ADB 里面输入 getprop 命令查看，如图所示为 RIL 版本号：

```
[gsm.version.ril-impl]: [WD_ZTE_RILV1.0.0B02_U6_20120320]
```

ZTE Confidential