LAMPIRAN A
SPESIFIKASI ALAT

1. Spesifikasi TP-Link TL-WR841ND AP Router N Series

FITUR PERANGKAT KERAS	
Tampilan	4 10/100Mbps LAN PORTS 1 10/100Mbps WAN PORT
Tombol	Quick Setup Security Button Reset Button
Catu Daya Eksternal	9VDC / 0.6A
Standar Nirkabel (Wireless)	IEEE 802.11n, IEEE 802.11g, IEEE 802.11b
Antena	2*5dBi Detachable Omni Directional Antenna (RP-SMA)
Dimensi (W x D x H)	7.6 x 5.1 x 1.3 in.(192 x 130 x 33 mm)
FITUR WIRELESS	
Frekuensi	2.4-2.4835GHz
Tingkat Signal	11n: Up to 300Mbps(dynamic) 11g: Up to 54Mbps(dynamic) 11b: Up to 11Mbps(dynamic)
EIRP	<20dBm(EIRP)
Penerimaan Sensitivitas	270M: -68dBm@10% PER 130M: -68dBm@10% PER 108M: -68dBm@10% PER 54M: -68dBm@10% PER 11M: -85dBm@8% PER 6M: -88dBm@10% PER 1M: -90dBm@8% PER
Fungsi Wireless	Enable/Disable Wireless Radio, WDS Bridge, WMM, Wireless Statistics
Keamanan Wireless	64/128/152-bit WEP / WPA / WPA2,WPA-PSK / WPA2-PSK
FITUR PERANGKAT LUNAK	
Tipe WAN	Dynamic IP/Static IP/PPPoE/ PPTP(Dual Access)/L2TP(Dual Access)/BigPond
DHCP	Server, Client, DHCP Client List, Address Reservation
Quality of Service	WMM, Bandwidth Control
Port Forwarding	Virtual Server,Port Triggering, UPnP, DMZ
Dynamic DNS	DynDns, Comexe, NO-IP
VPN Pass-Through	PPTP, L2TP, IPSec (ESP Head)
Access Control	Parental Control, Local Management Control, Host List, Access Schedule, Rule Management
Keamanan Firewall	DoS, SPI Firewall IP Address Filter/MAC Address Filter/Domain Filter IP and MAC Address Binding
Manajemen	Access Control Local Management Remote Management
Lainnya	
Sertifikasi	CE, FCC, RoHS
Isi Paket	Wireless N Rotuer TL-WR841ND 2 Detachable Omni Directional Antennas Power supply unit Resource CD Quick Installation Guide
Kebutuhan Sistem	Microsoft® Windows® 98SE, NT, 2000, XP, Vista™ or Windows 7, MAC® OS, NetWare®, UNIX® or Linux.
Lingkungan	Operating Temperature: 0℃~40℃ (32℉~104℉) Storage Temperature: -40℃~70℃ (-40℉~158℉) Operating Humidity: 10%~90% non-condensing Storage Humidity: 5%~90% non-condensing

2. Spesifikasi Samsung Galaxy W I18150

GENERAL	Network	2G GSM 850 / 900 / 1800 / 1900 3G HSDPA
	Tipe	Capacitive touchscreen, 16M colors
LAYAR	Ukuran	480 x 800 pixels, 3.7 inches (~252 ppi pixel density) - Multi-touch input method - Accelerometer sensor for UI auto-rotate - Touch-sensitive controls - Proximity sensor for auto turn-off
DIMENSI	Ukuran/Berat	115.5 x 59.8 x 11.5 mm / 109.9 g
AUDIO	Fitur	Vibration MP3, WAV ringtones
	Jack	3,5mm Jack Audio
	Speakerphone	Ya
MEMORY	Internal	512 MB RAM, 2 GB ROM
	Eksternal	microSD, up to 32GB
DATA	3G	HSDPA, 14.4 Mbps; HSUPA, 5.76 Mbps
	EDGE	Ya
	GPRS	Ya
	WLAN	Wi-Fi 802.11 b/g/n, Wi-Fi hotspot
	Bluetooth	Ya, v3.0 with A2DP
	Infrared	Tidak
	USB/Port	Ya, v2.0 microUSB
KAMERA	Primer	5 MP, 2592x1944 pixels, autofocus, LED flash - Geo-tagging, touch focus, face and smile detection
	Sekunder	Ya, VGA
	Video Record	Ya, 720p@30fps
BATERAI	Tipe	Standard battery, Li-Ion 1500 mAh
	Standby	Up to 570 h (2G) / Up to 420 h (3G)
	Talk Time	Up to 17 h 50 min (2G) / Up to 8 h 20 min (3G)
FITUR	OS	Android OS, v2.3 (Gingerbread)
	CPU	1.4 GHz Scorpion processor, Adreno 205 GPU, Qualcomm MSM8255T Snapdragon
	Browser	HTML
	GPS	Ya, with A-GPS support
	Messaging	SMS(threaded view), MMS, Email, Push Mail, IM, RSS
	Java	via Java MIDP emulator, Fitur tambahan: Stereo FM radio with RDS - SNS integration - Digital compass - Organizer - Image/video editor - Document editor (Word, Excel, PowerPoint, PDF) - Google Search, Maps, Gmail, YouTube, Calendar, Google Talk, Picasa integration - Adobe Flash support - Voice memo/dial/commands - Predictive text input (Swype)
	Multiple SIM	Tidak
FITUR LAIN	Video Player	- MP4/DivX/XviD/WMV/H.264/H.263 player
	MP3 Player	- MP3/WAV/eAAC+ player
	Audio Record	Ya
	TV	Tidak

3. Samsung Galaxy Tab2 7.0 P3100

GENERAL	Network	GSM 850 / 900 / 1800 / 1900 ,3G HSDPA 900 /1900 / 2100
LAYAR	Tipe	PLS LCD capacitive touchscreen, 16M colors
	Ukuran	600 x 1024 pixels, 7.0 inches (~170 ppi pixel density, Multitouch, TouchWiz UX UI
DIMENSI	Ukuran/Berat	193.7 x 122.4 x 10.5 mm / 344 g
AUDIO	Fitur	Vibration MP3, WAV ringtones
	Jack	3.5mm jack audio
	Speakerphone	Ya
MEMORY	Internal	16 GB storage, 1GB RAM
	Eksternal	microSD, up to 64 GB
DATA	3G	HSDPA, 21 Mbps; HSUPA, 5.76 Mbps
	EDGE	Ya
	GPRS	Ya
	WLAN	Wi-Fi 802.11 a/b/g/n, DLNA, Wi-Fi Direct, dual-band ,Wi-Fi hotspot
	Bluetooth	v3.0 with A2DP, HS
	Infrared	Tidak
	USB/Port	microUSB v2.0, USB On-the-go support
KAMERA	Primer	3.15 MP, 2048x1536 pixels, autofocus
	Sekunder	VGA
	Video Record	1080p@30fps
BATERAI	Tipe	Standard battery, Li-Ion 4000 mAh
	Standby	-
	Talk Time	-
FITUR	OS	Android OS, v4.0.3 (Ice Cream Sandwich)
	CPU	TI OMAP 4430, CPU Dual-core 1 GHz, GPU PowerVR SGX540
	Browser	HTML5, Adobe Flash
	GPS	Ya, A-GPS
	Messaging	SMS(threaded view), MMS, Email, Push Email, IM, RS
	Java	via Java MIDP emulator, fitur tambahan: - TV Out - SNS integration- Organizer - Image/video editor - Quickoffice HD editor/viewer - Google Search, Maps, Gmail, YouTube, Calendar, Google Talk, Picasa integration - Voice memo - Predictive text input (Swype)
FITUR LAIN	Multiple SIM	Tidak
	Video Player	MP4/DivX/Xvid/H.264/H.263/WMV player
	MP3 Player	MP3/WAV/eAAC+/WMA/AC3/Flac player
	Audio Record	Ya
	TV	Tidak

4. Smartfren Andromax U 4.5 LE

General	Network	GSM 850 / 900 / 1800 / 1900, CDMA 2000 1x, EVDO Rev A 800/1900 Mhz
Layar	Type	IPS LCD capacitive touchscreen, 16M colors
	Ukuran	4.5″ IPS Display with 540×960 pixels (256 ppi pixels density)
	Multitouch	Ya
	Proteksi	Tidak
Dimensi	Ukuran/Berat	130 x 66 x 9mm / 140 gram
Audio	Fitur	Vibration, MP3 Ringtones
	Jack	3,5 mm Jack Audio
	Speakerphone	Ya
Memory	Internal	4 GB, 1 GB RAM
	Eksternal	Micro SD Card Slot, up to 32 GB
Data	GPRS	Class B
	EDGE	Up to 236.8 kbps
	3G	EVDO Rev. A up to 3.1 Mbps
	WLAN	Wi-Fi 802.11 a/b/g/n, Wi-Fi hotspot
	Bluetooth	Ya, with A2DP
	NFC	Tidak
	USB/Port	microUSB v2.0
Kamera	Primer	8 MP, 3200?2400 piksel, autofocus, LED flash; Geo-tagging
	Sekunder	2 MP
	Video Record	Ya, Video HD 720p
Baterai	Tipe	Li-Ion 1800mAh
	Standby	-
	Talk Time	-
Fitur	OS	Android OS, v4.1.2 Jelly Bean
	CPU	Prosesor Qualcomm Snapdragon quad core 1.2 Ghz, Adreno 203 GPU
	Messaging	SMS (threaded view), MMS, Email, IM, Push Email
	Browser	HTML5
	Radio	FM radio
	GPS	Ya, with A-GPS
	Java	Via Java Emulator, Fitur tambahan: Stereo FM radio with RDS, Organizer (kalender, kalkulator, clock, voice recorder, Google Drive, Voice Dialer), Viki, VMS, Play Movies, Smartfren Mobile, Smart Pustakaoogle Maps, Google Play (Android Market), Gmail, Google Search, Gtalk,
	Fitur lainnya	Dual SIM CDMA-GSM, MP4/H.264/H.263 player, MP3/WAV/eAAC+ player & Audio Record

LAMPIRAN B

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="org.sipdroid.sipua"
    android:versionName="2.5 beta"
    android:versionCode="88"
    android:installLocation="auto">
          <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="10" android:maxSdkVersion="15"/>
          <supports-screens
         android:normalScreens="true"
         android:smallScreens="true"
         android:largeScreens="true"
         android:anyDensity="false" />
          <uses-feature android:name="android.hardware.bluetooth" android:required="false"/>
          <uses-feature android:name="android.hardware.location" android:required="false"/>
          <uses-feature android:name="android.hardware.location.gps" android:required="false"/>
          <uses-feature android:name="android.hardware.telephony" android:required="false"/>
          <uses-feature android:name="android.hardware.touchscreen" android:required="false"/>
          <uses-feature android:name="android.hardware.wifi" android:required="false"/>
          <uses-feature android:name="android.hardware.camera" android:required="false"/>
          <application android:icon="@drawable/asterisk" android:label="@string/app_name">
        <activity android:name="org.sipdroid.sipua.ui.Sipdroid" android:label="@string/app_name"
android:launchMode="singleInstance"
          android:configChanges="orientation|keyboardHidden">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
        </activity>
        <activity android:name="org.sipdroid.sipua.ui.PSTN" android:label="@string/pstn_name"
           android:icon="@drawable/ic_launcher_phone">
           <intent-filter>
               <action android:name="android.intent.action.SENDTO" />
               <category android:name="android.intent.category.DEFAULT" />
               <data android:scheme="sms" />
               <data android:scheme="smsto" />
           </intent-filter>
        </activity>
        <activity android:name="org.sipdroid.sipua.ui.SIP" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.SENDTO" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="sms" />
                <data android:scheme="smsto" />
            </intent-filter>

        </activity>
        <activity android:name="org.sipdroid.sipua.ui.AutoAnswer" android:label="@string/app_name"/>
        <activity android:name="org.sipdroid.sipua.ui.ChangeAccount" android:label="@string/app_name"/>
        <activity android:name="org.sipdroid.sipua.ui.SIPUri" android:label="@string/app_name"
                    android:theme="@android:style/Theme.Dialog">
            <intent-filter>
                <action android:name="android.intent.action.CALL" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="sip" />
                <data android:scheme="sipdroid" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.SENDTO" />
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:scheme="imto" />
                <data android:scheme="sip" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.CALL_PRIVILEGED" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="sip" />
            </intent-filter>
        </activity>
        <activity android:name="org.sipdroid.sipua.ui.Activity2" android:label="@string/app_name"
           android:excludeFromRecents="true" android:taskAffinity=""/>
        <activity android:name="org.sipdroid.sipua.ui.Settings" android:label="@string/app_name"
           android:configChanges="orientation|keyboardHidden">
           android:excludeFromRecents="true" android:taskAffinity="">
        </activity>
        <activity
              android:name="org.sipdroid.codecs.Codecs$CodecSettings"
              android:label="@string/app_name" >
        </activity>
        <activity android:name="org.sipdroid.sipua.ui.VideoCamera" android:label="@string/menu_video"
                   android:excludeFromRecents="true" android:taskAffinity=""
                              android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen"
                   android:screenOrientation="landscape"
                   android:clearTaskOnLaunch="true"
                   android:configChanges="orientation|keyboardHidden">
        </activity>
        <activity android:name="org.sipdroid.sipua.ui.InCallScreen" android:label="@string/app_name"
                   android:excludeFromRecents="true" android:taskAffinity=""
                   android:launchMode="singleInstance">
        </activity>
```

```
            <receiver android:name="org.sipdroid.sipua.ui.OneShotAlarm"/>
            <receiver android:name="org.sipdroid.sipua.ui.OneShotAlarm2"/>
            <receiver android:name="org.sipdroid.sipua.ui.LoopAlarm"/>
            <receiver android:name="org.sipdroid.sipua.ui.OwnWifi"/>
            <receiver android:name="org.sipdroid.sipua.ui.OneShotLocation"/>
            <receiver android:name="org.sipdroid.sipua.ui.LoopLocation"/>
            <receiver android:name="org.sipdroid.sipua.ui.Caller">
                <intent-filter android:priority="-1">
                   <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
                </intent-filter>
            </receiver>
       <receiver android:name="org.sipdroid.sipua.ui.Receiver" android:enabled="true">
                   <intent-filter>
                   <action android:name="android.intent.action.BOOT_COMPLETED" />
                   <action android:name="android.intent.action.EXTERNAL_APPLICATIONS_AVAILABLE" />
                   <action android:name="android.intent.action.EXTERNAL_APPLICATIONS_UNAVAILABLE" />
                   </intent-filter>
                   <intent-filter>
                   <action android:name="android.intent.action.PACKAGE_REPLACED" />
                   <data android:scheme="package" />
                   <data android:path="org.sipdroid.sipua" />
                   </intent-filter>
            </receiver>
          <service android:name="org.sipdroid.sipua.ui.RegisterService" />
    </application>
          <uses-permission android:name="android.permission.INTERNET"></uses-permission>
          <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"></uses-permission>
          <uses-permission android:name="android.permission.RECORD_AUDIO"></uses-permission>
          <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"></uses-permission>
          <uses-permission android:name="android.permission.WRITE_SETTINGS"></uses-permission>
          <uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
          <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
          <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-permission>
          <uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
          <uses-permission android:name="android.permission.WRITE_CONTACTS"></uses-permission>
          <uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
          <uses-permission android:name="android.permission.WAKE_LOCK"></uses-permission>
          <uses-permission android:name="android.permission.DISABLE_KEYGUARD"></uses-permission>
          <uses-permission android:name="android.permission.CAMERA"></uses-permission>
          <uses-permission android:name="android.permission.VIBRATE" ></uses-permission>
          <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" ></uses-permission>
          <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" ></uses-permission>
          <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" ></uses-permission>
          <uses-permission android:name="android.permission.BLUETOOTH" ></uses-permission>
          <uses-permission android:name="android.permission.GET_ACCOUNTS" ></uses-permission>
          <uses-permission android:name="android.permission.BROADCAST_STICKY" ></uses-permission>
</manifest>
```

- **Folder src**
- **/Sipdroid/src/org/sipdroid/codecs**

G711.java

```
package org.sipdroid.codecs;

/**
 * G.711 codec. This class provides methods for u-law, A-law and linear PCM
 * conversions.
 */
public class G711 {
          /*
           * Copyright 1992 by Jutta Degener and Carsten Bormann, Technische
           * Universitaet Berlin.  See the accompanying file "COPYRIGHT" for
           * details.  THERE IS ABSOLUTELY NO WARRANTY FOR THIS SOFTWARE.
           */

          static final short[] a2s = new short[256];
          static final int[] _a2s = {

           60032, 60288, 59520, 59776, 61056, 61312, 60544, 60800,
           57984, 58240, 57472, 57728, 59008, 59264, 58496, 58752,
           62784, 62912, 62528, 62656, 63296, 63424, 63040, 63168,
           61760, 61888, 61504, 61632, 62272, 62400, 62016, 62144,
           43520, 44544, 41472, 42496, 47616, 48640, 45568, 46592,
           35328, 36352, 33280, 34304, 39424, 40448, 37376, 38400,
           54528, 55040, 53504, 54016, 56576, 57088, 55552, 56064,
           50432, 50944, 49408, 49920, 52480, 52992, 51456, 51968,
           65192, 65208, 65160, 65176, 65256, 65272, 65224, 65240,
           65064, 65080, 65032, 65048, 65128, 65144, 65096, 65112,
           65448, 65464, 65416, 65432, 65512, 65528, 65480, 65496,
           65320, 65336, 65288, 65304, 65384, 65400, 65352, 65368,
           64160, 64224, 64032, 64096, 64416, 64480, 64288, 64352,
           63648, 63712, 63520, 63584, 63904, 63968, 63776, 63840,
           64848, 64880, 64784, 64816, 64976, 65008, 64912, 64944,
           64592, 64624, 64528, 64560, 64720, 64752, 64656, 64688,
            5504,  5248,  6016,  5760,  4480,  4224,  4992,  4736,
            7552,  7296,  8064,  7808,  6528,  6272,  7040,  6784,
            2752,  2624,  3008,  2880,  2240,  2112,  2496,  2368,
            3776,  3648,  4032,  3904,  3264,  3136,  3520,  3392,
           22016, 20992, 24064, 23040, 17920, 16896, 19968, 18944,
           30208, 29184, 32256, 31232, 26112, 25088, 28160, 27136,
```

```
 11008, 10496, 12032, 11520,  8960,  8448,  9984,  9472,
 15104, 14592, 16128, 15616, 13056, 12544, 14080, 13568,
   344,   328,   376,   360,   280,   264,   312,   296,
   472,   456,   504,   488,   408,   392,   440,   424,
    88,    72,   120,   104,    24,     8,    56,    40,
   216,   200,   248,   232,   152,   136,   184,   168,
  1376,  1312,  1504,  1440,  1120,  1056,  1248,  1184,
  1888,  1824,  2016,  1952,  1632,  1568,  1760,  1696,
   688,   656,   752,   720,   560,   528,   624,   592,
   944,   912,  1008,   976,   816,   784,   880,   848

};

static final byte[] s2a = new byte[65536];
static final int[] _s2a = {

          213,212,215,214,209,208,211,210,221,220,223,222,217,216,219,218,
          197,196,199,198,193,192,195,194,205,204,207,206,201,200,203,202,
          245,245,244,244,247,247,246,246,241,241,240,240,243,243,242,242,
          253,253,252,252,255,255,254,254,249,249,248,248,251,251,250,250,
          229,229,229,229,228,228,228,228,231,231,231,231,230,230,230,230,
          225,225,225,225,224,224,224,224,227,227,227,227,226,226,226,226,
          237,237,237,237,236,236,236,236,239,239,239,239,238,238,238,238,
          233,233,233,233,232,232,232,232,235,235,235,235,234,234,234,234,
          149,149,149,149,149,149,149,149,148,148,148,148,148,148,148,148,
          151,151,151,151,151,151,151,151,150,150,150,150,150,150,150,150,
          145,145,145,145,145,145,145,145,144,144,144,144,144,144,144,144,
          147,147,147,147,147,147,147,147,146,146,146,146,146,146,146,146,
          157,157,157,157,157,157,157,157,156,156,156,156,156,156,156,156,
          159,159,159,159,159,159,159,159,158,158,158,158,158,158,158,158,
          153,153,153,153,153,153,153,153,152,152,152,152,152,152,152,152,
          155,155,155,155,155,155,155,155,154,154,154,154,154,154,154,154,
          133,133,133,133,133,133,133,133,133,133,133,133,133,133,133,133,
          132,132,132,132,132,132,132,132,132,132,132,132,132,132,132,132,
          135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,
          134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,
          129,129,129,129,129,129,129,129,129,129,129,129,129,129,129,129,
          128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,
          131,131,131,131,131,131,131,131,131,131,131,131,131,131,131,131,
          130,130,130,130,130,130,130,130,130,130,130,130,130,130,130,130,
          141,141,141,141,141,141,141,141,141,141,141,141,141,141,141,141,
          140,140,140,140,140,140,140,140,140,140,140,140,140,140,140,140,
          143,143,143,143,143,143,143,143,143,143,143,143,143,143,143,143,
          142,142,142,142,142,142,142,142,142,142,142,142,142,142,142,142,
          137,137,137,137,137,137,137,137,137,137,137,137,137,137,137,137,
          136,136,136,136,136,136,136,136,136,136,136,136,136,136,136,136,
          139,139,139,139,139,139,139,139,139,139,139,139,139,139,139,139,
          138,138,138,138,138,138,138,138,138,138,138,138,138,138,138,138,
          181,181,181,181,181,181,181,181,181,181,181,181,181,181,181,181,
          181,181,181,181,181,181,181,181,181,181,181,181,181,181,181,181,
          180,180,180,180,180,180,180,180,180,180,180,180,180,180,180,180,
          180,180,180,180,180,180,180,180,180,180,180,180,180,180,180,180,
          183,183,183,183,183,183,183,183,183,183,183,183,183,183,183,183,
          183,183,183,183,183,183,183,183,183,183,183,183,183,183,183,183,
          182,182,182,182,182,182,182,182,182,182,182,182,182,182,182,182,
          182,182,182,182,182,182,182,182,182,182,182,182,182,182,182,182,
          177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,
          177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,177,
          176,176,176,176,176,176,176,176,176,176,176,176,176,176,176,176,
          176,176,176,176,176,176,176,176,176,176,176,176,176,176,176,176,
          179,179,179,179,179,179,179,179,179,179,179,179,179,179,179,179,
          179,179,179,179,179,179,179,179,179,179,179,179,179,179,179,179,
          178,178,178,178,178,178,178,178,178,178,178,178,178,178,178,178,
          178,178,178,178,178,178,178,178,178,178,178,178,178,178,178,178,
          189,189,189,189,189,189,189,189,189,189,189,189,189,189,189,189,
          189,189,189,189,189,189,189,189,189,189,189,189,189,189,189,189,
          188,188,188,188,188,188,188,188,188,188,188,188,188,188,188,188,
          188,188,188,188,188,188,188,188,188,188,188,188,188,188,188,188,
          191,191,191,191,191,191,191,191,191,191,191,191,191,191,191,191,
          191,191,191,191,191,191,191,191,191,191,191,191,191,191,191,191,
          190,190,190,190,190,190,190,190,190,190,190,190,190,190,190,190,
          190,190,190,190,190,190,190,190,190,190,190,190,190,190,190,190,
          185,185,185,185,185,185,185,185,185,185,185,185,185,185,185,185,
          185,185,185,185,185,185,185,185,185,185,185,185,185,185,185,185,
          184,184,184,184,184,184,184,184,184,184,184,184,184,184,184,184,
          184,184,184,184,184,184,184,184,184,184,184,184,184,184,184,184,
          187,187,187,187,187,187,187,187,187,187,187,187,187,187,187,187,
          187,187,187,187,187,187,187,187,187,187,187,187,187,187,187,187,
          186,186,186,186,186,186,186,186,186,186,186,186,186,186,186,186,
          186,186,186,186,186,186,186,186,186,186,186,186,186,186,186,186,
          165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,
          165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,
          165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,
          165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,165,
          164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,
          164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,
          164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,
          164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,164,
          167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,
          167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,
          167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,
          167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,167,
          166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,
          166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,
          166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,
```

166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,166,
161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,
161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,
161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,
161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,161,
160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,
160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,
160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,
160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,160,
163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,
163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,
163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,
163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,
162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,
162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,
162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,
162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,162,
173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,
173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,
173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,
173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,173,
172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,
172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,
172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,
172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,172,
175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,
175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,
175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,
175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,175,
174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,
174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,
174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,
174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,174,
169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,
169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,
169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,
169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,169,
168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,
168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,
168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,
168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,168,
171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,
171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,
171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,
171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,171,
170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,
170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,
170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,
170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,170,
 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42,
 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42,
 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42,
 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42,
 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43,
 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43,
 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43,
 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43, 43,
 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41,
 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41,
 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41,
 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41, 41,
 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46,
 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46,
 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46,
 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, 46,
 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47, 47,
 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44,
 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44,
 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44,
 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 44,
 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45,
 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45,
 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45,
 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45,
 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34,
 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34,
 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34,
 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34,
 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35,
 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35,
 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35,
 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35,
 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,

```
            33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
            33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
            38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38,
            38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38,
            38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38,
            38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38,
            39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
            39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
            39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
            39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
            36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36,
            36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36,
            36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36,
            36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36,
            37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37,
            37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37,
            37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37,
            37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37, 37,
            58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58,
            58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58, 58,
            59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59,
            59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59, 59,
            56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56,
            56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56, 56,
            57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57,
            57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57,
            62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
            62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
            63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63,
            63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63, 63,
            60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
            60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
            61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61,
            61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61,
            50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
            50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50,
            51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
            51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
            48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48,
            48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48, 48,
            49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
            49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
            54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54,
            54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54, 54,
            55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55,
            55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55,
            52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
            52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52,
            53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
            53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53, 53,
            10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
            11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
             8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
             9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
            14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
            15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
            12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
            13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
             2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
             3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
             0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
             1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
             6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
             7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
             4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
             5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
            26, 26, 26, 26, 26, 26, 26, 26, 27, 27, 27, 27, 27, 27, 27, 27,
            24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25, 25, 25, 25,
            30, 30, 30, 30, 30, 30, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31,
            28, 28, 28, 28, 28, 28, 28, 28, 29, 29, 29, 29, 29, 29, 29, 29,
            18, 18, 18, 18, 18, 18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19,
            16, 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 17, 17, 17,
            22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23,
            20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21,
           106,106,106,106,107,107,107,107,104,104,104,104,105,105,105,105,
           110,110,110,110,111,111,111,111,108,108,108,108,109,109,109,109,
            98, 98, 98, 98, 99, 99, 99, 99, 96, 96, 96, 96, 97, 97, 97, 97,
           102,102,102,102,103,103,103,103,100,100,100,100,101,101,101,101,
           122,122,123,123,120,120,121,121,126,126,127,127,124,124,125,125,
           114,114,115,115,112,112,113,113,118,118,119,119,116,116,117,117,
            74, 75, 72, 73, 78, 79, 76, 77, 66, 67, 64, 65, 70, 71, 68, 69,
            90, 91, 88, 89, 94, 95, 92, 93, 82, 83, 80, 81, 86, 87, 84, 85
};

//change G711 ulaw start
static final int _u2a[] = {                                             /* u- to A-law conversions */
          1,                  1,                  2,                  2,                  3,
3,                  4,                  4,
          5,                  5,                  6,                  6,                  7,
7,                  8,                  8,
          9,                 10,                 11,                 12,                 13,
         14,                 15,                 16,
         17,                 18,                 19,                 20,                 21,
         22,                 23,                 24,
         25,                 27,                 29,                 31,                 33,
         34,                 35,                 36,
         37,                 38,                 39,                 40,                 41,
         42,                 43,                 44,
         46,                 48,                 49,                 50,                 51,
         52,                 53,                 54,
```

```
                55,     56,     57,     58,     59,     60,     61,     62,
                64,     65,     66,     67,     68,     69,     70,     71,
                72,     73,     74,     75,     76,     77,     78,     79,
                81,     82,     83,     84,     85,     86,     87,     88,
                89,     90,     91,     92,     93,     94,     95,     96,
                97,     98,     99,     100,    101,    102,    103,    104,
                105,    106,    107,    108,    109,    110,    111,    112,
                113,    114,    115,    116,    117,    118,    119,    120,
                121,    122,    123,    124,    125,    126,    127,    128};

        static final int _a2u[] = {                                     /* A- to u-law conversions */
                1,              3,              5,              7,              9,
        11,     13,     15,
                16,     17,     18,     19,     20,     21,     22,     23,
                24,     25,     26,     27,     28,     29,     30,     31,
                32,     32,     33,     33,     34,     34,     35,     35,
                36,     37,     38,     39,     40,     41,     42,     43,
                44,     45,     46,     47,     48,     48,     49,     49,
                50,     51,     52,     53,     54,     55,     56,     57,
                58,     59,     60,     61,     62,     63,     64,     64,
                65,     66,     67,     68,     69,     70,     71,     72,
                73,     74,     75,     76,     77,     78,     79,     80,
                80,     81,     82,     83,     84,     85,     86,     87,
                88,     89,     90,     91,     92,     93,     94,     95,
                96,     97,     98,     99,     100,    101,    102,    103,
                104,    105,    106,    107,    108,    109,    110,    111,
                112,    113,    114,    115,    116,    117,    118,    119,
                120,    121,    122,    123,    124,    125,    126,    127};

        //change end

        public static void init() {
        }

        static {
                        int i;
                        for (i = 0; i < 256; i++)
                                a2s[i] = (short)_a2s[i];
                        for (i = 0; i < 65536; i++)
                                s2a[i] = (byte)_s2a[i >> 4];
        }

        public static void alaw2linear(byte alaw[],short lin[],int frames) {
                int i;
                for (i = 0; i < frames; i++)
                        lin[i] = a2s[alaw[i+12] & 0xff];
        }

        public static void alaw2linear(byte alaw[],short lin[],int frames,int mu) {
                int i;
                for (i = 0; i < frames; i++)
                        lin[i] = a2s[alaw[i/mu+12] & 0xff];
        }

        public static void linear2alaw(short lin[],int offset,byte alaw[],int frames) {
                int i;
                for (i = 0; i < frames; i++)
                        alaw[i+12] = s2a[lin[i+offset] & 0xffff];
        }

        //change g711 ulaw start
   protected static int alaw2ulaw(int aval)
   {  aval&=0xff;
           return ((aval & 0x80)!=0)? (0xFF^_a2u[aval^0xD5]) : (0x7F^_a2u[aval^0x55]);
   }

   protected static int ulaw2alaw(int uval)
   {  uval&=0xff;
           return ((uval&0x80)!=0)? (0xD5^(_u2a[0xFF^uval]-1)) : (0x55^(_u2a[0x7F^uval]-1));
   }

        public static void ulaw2linear(byte ulaw[],short lin[],int frames) {
                int i;
                for (i = 0; i < frames; i++)
                        lin[i] = a2s[ulaw2alaw(ulaw[i+12] & 0xff)];
        }
        public static void linear2ulaw(short lin[],int offset,byte ulaw[],int frames) {
                int i;
                for (i = 0; i < frames; i++)
                        ulaw[i+12] = (byte)alaw2ulaw(s2a[lin[i+offset] & 0xffff]);
        }
        //change end
}
```

alaw.java

```
package org.sipdroid.codecs;


class alaw extends CodecBase implements Codec {
        alaw() {
```

```
                CODEC_NAME = "PCMA";
                CODEC_USER_NAME = "PCMA";
                CODEC_DESCRIPTION = "64kbit";
                CODEC_NUMBER = 8;
                CODEC_DEFAULT_SETTING = "wlanor3g";

                load();
        }

        public void init() {
                G711.init();
        }

        public int decode(byte enc[], short lin[], int frames) {
                G711.alaw2linear(enc, lin, frames);

                return frames;
        }

        public int encode(short lin[], int offset, byte enc[], int frames) {
                G711.linear2alaw(lin, offset, enc, frames);

                return frames;
        }

        public void close() {
        }
}
```

Ulaw.java

```
package org.sipdroid.codecs;


class ulaw extends CodecBase implements Codec {
        ulaw() {
                CODEC_NAME = "PCMU";
                CODEC_USER_NAME = "PCMU";
                CODEC_DESCRIPTION = "64kbit";
                CODEC_NUMBER = 0;
                CODEC_DEFAULT_SETTING = "wlanor3g";

                load();
        }

        public void init() {
                G711.init();
        }

        public int decode(byte enc[], short lin[], int frames) {
                G711.ulaw2linear(enc, lin, frames);

                return frames;
        }

        public int encode(short lin[], int offset, byte enc[], int frames) {
                G711.linear2ulaw(lin, offset, enc, frames);

                return frames;
        }

        public void close() {
        }
}
```

Codec.java

```
package org.sipdroid.codecs;

import android.preference.ListPreference;

/**
 * Represents the basic interface to the Codec classes All codecs need
 * to implement basic encode and decode capability Codecs which
 * inherit from {@link CodecBase} only need to implement encode,
 * decode and init
 */
public interface Codec {
        /**
         * Decode a linear pcm audio stream
         *
         * @param encoded The encoded audio stream
         *
         * @param lin The linear pcm audio frame buffer in which to place the decoded stream
         *
         * @param size The size of the encoded frame
         *
```

```
 * @returns The size of the decoded frame
 */
int decode(byte encoded[], short lin[], int size);

/**
 * Encode a linear pcm audio stream
 *
 * @param lin The linear stream to encode
 *
 * @param offset The offset into the linear stream to begin
 *
 * @param encoded The buffer to place the encoded stream
 *
 * @param size the size of the linear pcm stream (in words)
 *
 * @returns the length (in bytes) of the encoded stream
 */
int encode(short lin[], int offset, byte alaw[], int frames);

/**
 * The sampling rate for this particular codec
 */
int samp_rate();

/**
 * The audio frame size for this particular codec
 */
int frame_size();

/**
 * Optionally used to initiallize the codec before any
 * encoding or decoding
 */
void init();
void update();

/**
 * Optionally used to free any resources allocated in init
 * after encoding or decoding is complete
 */
void close();

/**
 * (implemented by {@link CodecBase}
 * <p>
 * checks to see if the user has enabled the codec.
 *
 * @returns true if the codec can be used
 */
boolean isEnabled();

/**
 * (implemented by {@link CodecBase}
 * <p>
 * Checks to see if the binary library associated with the
 * codec (if any) loaded OK.
 *
 * @returns true if the codec loaded properly
 */
boolean isLoaded();
boolean isFailed();
void fail();
boolean isValid();

/**
 * (implemented by {@link CodecBase}
 *
 * @returns The user friendly string for the codec (should
 * include both the name and the bandwidth
 */
String getTitle();


/**
 * (implemented by {@link CodecBase}
 *
 * @returns The RTP assigned name string for the codec
 */
String name();
String key();
String getValue();

/**
 * (implemented by {@link CodecBase}
 *
 * @returns The commonly used name for the codec.
 */
String userName();

/**
 * (implemented by {@link CodecBase}
 *
 * @returns The RTP assigned number for the codec
 */
int number();
```

```
	/**
	 * (implemented by {@link CodecBase}
	 *
	 * @param l The list preference controlling this Codec
	 *
	 * Used to add listeners for preference changes and update
	 * the codec parameters accordingly.
	 */
	void setListPreference(ListPreference l);
}
```

Codecs.java

```
package org.sipdroid.codecs;
import java.util.HashMap;
import java.util.Vector;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.ui.Receiver;
import org.sipdroid.sipua.ui.Settings;
import org.zoolu.sdp.MediaField;
import org.zoolu.sdp.SessionDescriptor;
import org.zoolu.sdp.AttributeField;

import android.content.Context;
import android.content.res.Resources;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.Preference;
import android.preference.PreferenceActivity;
import android.preference.ListPreference;
import android.preference.PreferenceManager;
import android.preference.PreferenceScreen;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;

public class Codecs {
	private static final Vector<Codec> codecs = new Vector<Codec>() {{
//				add(new G722());
//				add(new SILK24());		save space (until a common library for all bitrates
gets available?)
//				add(new SILK16());
//				add(new SILK8());
				add(new alaw());
				add(new ulaw());
//				add(new Speex());
//				add(new GSM());
//				add(new BV16());
			}};
	private static final HashMap<Integer, Codec> codecsNumbers;
	private static final HashMap<String, Codec> codecsNames;

	static {
			final int size = codecs.size();
			codecsNumbers = new HashMap<Integer, Codec>(size);
			codecsNames = new HashMap<String, Codec>(size);

			for (Codec c : codecs) {
				codecsNames.put(c.name(), c);
				codecsNumbers.put(c.number(), c);
			}

			SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(Receiver.mContext);
			String prefs = sp.getString(Settings.PREF_CODECS, Settings.DEFAULT_CODECS);
			if (prefs == null) {
				String v = "";
				SharedPreferences.Editor e = sp.edit();

				for (Codec c : codecs)
					v = v + c.number() + " ";
				e.putString(Settings.PREF_CODECS, v);
				e.commit();
			} else {
				String[] vals = prefs.split(" ");
				for (String v: vals) {
					try {
						int i = Integer.parseInt(v);
						Codec c = codecsNumbers.get(i);
						/* moves the codec to the end
						 * of the list so we end up
						 * with the new codecs (if
						 * any) at the top and the
						 * remaining ones ordered
						 * according to the user */
						if (c != null) {
							codecs.remove(c);
							codecs.add(c);
						}
					} catch (Exception e) {
						// do nothing (expecting
						// NumberFormatException and
```

```
                                                  // indexnot found
                                        }
                                }
                        }
                }

                public static Codec get(int key) {
                        return codecsNumbers.get(key);
                }

                public static Codec getName(String name) {
                        return codecsNames.get(name);
                }

                public static void check() {
                        HashMap<String, String> old = new HashMap<String, String>(codecs.size());

                        for(Codec c : codecs) {
                                c.update();
                                old.put(c.name(), c.getValue());
                                if (!c.isLoaded()) {
                                        SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(Receiver.mContext);
                                        SharedPreferences.Editor e = sp.edit();

                                        e.putString(c.key(), "never");
                                        e.commit();
                                }
                        }

                        for(Codec c : codecs)
                                if (!old.get(c.name()).equals("never")) {
                                        c.init();
                                        if (c.isLoaded()) {
                                                SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(Receiver.mContext);
                                                SharedPreferences.Editor e = sp.edit();

                                                e.putString(c.key(), old.get(c.name()));
                                                e.commit();
                                                c.init();
                                        } else
                                                c.fail();
                                }
                }

                private static void addPreferences(PreferenceScreen ps) {
                        Context cx = ps.getContext();
                        Resources r = cx.getResources();
                        ps.setOrderingAsAdded(true);

                        for(Codec c : codecs) {
                                ListPreference l = new ListPreference(cx);
                                l.setEntries(r.getStringArray(R.array.compression_display_values));
                                l.setEntryValues(r.getStringArray(R.array.compression_values));
                                l.setKey(c.key());
                                l.setPersistent(true);
                                l.setEnabled(!c.isFailed());
                                c.setListPreference(l);
                                if (c.number() == 9)
                                        if (ps.getSharedPreferences().getString(Settings.PREF_SERVER,
Settings.DEFAULT_SERVER).equals(Settings.DEFAULT_SERVER))
                                                l.setSummary(l.getEntry()+"
("+r.getString(R.string.settings_improve2)+")");
                                        else
                                                l.setSummary(l.getEntry()+"
("+r.getString(R.string.settings_hdvoice)+")");
                                else
                                        l.setSummary(l.getEntry());
                                l.setTitle(c.getTitle());
                                ps.addPreference(l);
                        }
                }

                public static int[] getCodecs() {
                        Vector<Integer> v = new Vector<Integer>(codecs.size());

                        for (Codec c : codecs) {
                                c.update();
                                if (!c.isValid())
                                        continue;
                                v.add(c.number());
                        }
                        int i[] = new int[v.size()];
                        for (int j = 0; j < i.length; j++)
                                i[j] = v.elementAt(j);
                        return i;
                }

                public static class Map {
                        public int number;
                        public Codec codec;
                        Vector<Integer> numbers;
                        Vector<Codec> codecs;
```

```
                Map(int n, Codec c, Vector<Integer> ns, Vector<Codec> cs) {
                        number = n;
                        codec = c;
                        numbers = ns;
                        codecs = cs;
                }

                public boolean change(int n) {
                        int i = numbers.indexOf(n);

                        if (i >= 0 && codecs.elementAt(i) != null) {
                                codec.close();
                                number = n;
                                codec = codecs.elementAt(i);
                                return true;
                        }
                        return false;
                }

                public String toString() {
                        return "Codecs.Map { " + number + ": " + codec + "}";
                }
        };

        public static Map getCodec(SessionDescriptor offers) {
                MediaField m = offers.getMediaDescriptor("audio").getMedia();
                if (m==null)
                        return null;

                String proto = m.getTransport();
                //see http://tools.ietf.org/html/rfc4566#page-22, paragraph 5.14, <fmt> description
                if ( proto.equals("RTP/AVP") || proto.equals("RTP/SAVP") ) {
                        Vector<String> formats = m.getFormatList();
                        Vector<String> names = new Vector<String>(formats.size());
                        Vector<Integer> numbers = new Vector<Integer>(formats.size());
                        Vector<Codec> codecmap = new Vector<Codec>(formats.size());

                        //add all avail formats with empty names
                        for (String fmt : formats) {
                                try {
                                        int number = Integer.parseInt(fmt);
                                        numbers.add(number);
                                        names.add("");
                                        codecmap.add(null);
                                } catch (NumberFormatException e) {
                                        // continue ... remote sent bogus rtp setting
                                }
                        };

                        //if we have attrs for format -> set name
                        Vector<AttributeField> attrs =
offers.getMediaDescriptor("audio").getAttributes("rtpmap");
                        for (AttributeField a : attrs) {
                                String s = a.getValue();
                                // skip over "rtpmap:"
                                s = s.substring(7, s.indexOf("/"));
                                int i = s.indexOf(" ");
                                try {
                                        String name = s.substring(i + 1);
                                        int number = Integer.parseInt(s.substring(0, i));
                                        int index = numbers.indexOf(number);
                                        if (index >=0)
                                                names.set(index, name.toLowerCase());
                                } catch (NumberFormatException e) {
                                        // continue ... remote sent bogus rtp setting
                                }
                        }

                        Codec codec = null;
                        int index = formats.size() + 1;

                        for (Codec c : codecs) {
                                c.update();
                                if (!c.isValid())
                                        continue;

                                //search current codec in offers by name
                                int i = names.indexOf(c.userName().toLowerCase());
                                if (i >= 0) {
                                        codecmap.set(i, c);
                                        if ( (codec==null) || (i < index) ) {
                                                codec = c;
                                                index = i;
                                                continue;
                                        }
                                }

                                //search current codec in offers by number
                                i = numbers.indexOf(c.number());
                                if (i >= 0) {
                                                if ( names.elementAt(i).equals("")) {
                                                        codecmap.set(i, c);
                                                        if ( (codec==null) || (i < index) )  {
                                                                //fmt number has no attr with
name
```

```
                                                                                codec = c;
                                                                                index = i;
                                                                                continue;
                                                                        }
                                                                }
                                                        }
                                        }
                                        if (codec!=null)
                                                return new Map(numbers.elementAt(index), codec, numbers, codecmap);
                                        else
                                                // no codec found ... we can't talk
                                                return null;
                                } else
                                        /*formats of other protocols not supported yet*/
                                        return null;
                        }

                        public static class CodecSettings extends PreferenceActivity {

                                private static final int MENU_UP = 0;
                                private static final int MENU_DOWN = 1;

                                @Override
                                public void onCreate(Bundle savedInstanceState) {
                                        super.onCreate(savedInstanceState);

                                        addPreferencesFromResource(R.xml.codec_settings);

                                        // for long-press gesture on a profile preference
                                        registerForContextMenu(getListView());

                                        addPreferences(getPreferenceScreen());
                                }

                                @Override
                                public void onCreateContextMenu(ContextMenu menu, View v,
                                                                        ContextMenuInfo menuInfo) {
                                        super.onCreateContextMenu(menu, v, menuInfo);

                                        menu.setHeaderTitle(R.string.codecs_move);
                                        menu.add(Menu.NONE, MENU_UP, 0,
                                                    R.string.codecs_move_up);
                                        menu.add(Menu.NONE, MENU_DOWN, 0,
                                                    R.string.codecs_move_down);
                                }

                                @Override
                                public boolean onContextItemSelected(MenuItem item) {

                                        int posn = (int)((AdapterContextMenuInfo)item.getMenuInfo()).position;
                                        Codec c = codecs.elementAt(posn);
                                        if (item.getItemId() == MENU_UP) {
                                                if (posn == 0)
                                                        return super.onContextItemSelected(item);
                                                Codec tmp = codecs.elementAt(posn - 1);
                                                codecs.set(posn - 1, c);
                                                codecs.set(posn, tmp);
                                        } else if (item.getItemId() == MENU_DOWN) {
                                                if (posn == codecs.size() - 1)
                                                        return super.onContextItemSelected(item);
                                                Codec tmp = codecs.elementAt(posn + 1);
                                                codecs.set(posn + 1, c);
                                                codecs.set(posn, tmp);
                                        }
                                        PreferenceScreen ps = getPreferenceScreen();
                                        SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(Receiver.mContext);
                                        String v = "";
                                        SharedPreferences.Editor e = sp.edit();

                                        for (Codec d : codecs)
                                                v = v + d.number() + " ";
                                        e.putString(Settings.PREF_CODECS, v);
                                        e.commit();
                                        ps.removeAll();
                                        addPreferences(ps);
                                        return super.onContextItemSelected(item);
                                }

                                @Override
                                public boolean onPreferenceTreeClick(PreferenceScreen ps, Preference p) {
                                        ListPreference l = (ListPreference) p;
                                        for (Codec c : codecs)
                                                if (c.key().equals(l.getKey())) {
                                                        c.init();
                                                        if (!c.isLoaded()) {
                                                                l.setValue("never");
                                                                c.fail();
                                                                l.setEnabled(false);
                                                                l.setSummary(l.getEntry());
                                                                if (l.getDialog() != null)
                                                                        l.getDialog().dismiss();
                                                        }
                                                }
                                        return super.onPreferenceTreeClick(ps,p);
```

```
                    }

                    @Override
                    public void onDestroy() {
                              super.onDestroy();
                              unregisterForContextMenu(getListView());
                    }
          }
}
```

➢ /Sipdroid/src/org/sipdroid/media

JAudioLauncher.java

```
package org.sipdroid.media;

import org.sipdroid.codecs.Codecs;
import org.sipdroid.net.SipdroidSocket;
import org.sipdroid.sipua.ui.Receiver;
import org.sipdroid.sipua.ui.Sipdroid;
import org.zoolu.sip.provider.SipStack;
import org.zoolu.tools.Log;
import org.zoolu.tools.LogLevel;

import android.preference.PreferenceManager;

/** Audio launcher based on javax.sound  */
public class JAudioLauncher implements MediaLauncher
{
   /** Event logger. */
   Log log=null;

   /** Sample rate [bytes] */
   int sample_rate=8000;
   /** Sample size [bytes] */
   int sample_size=1;
   /** Frame size [bytes] */
   int frame_size=160;
   /** Frame rate [frames per second] */
   int frame_rate=50; //=sample_rate/(frame_size/sample_size);
   boolean signed=false;
   boolean big_endian=false;

   //String filename="audio.wav";

   /** Test tone */
   public static final String TONE="TONE";

   /** Test tone frequency [Hz] */
   public static int tone_freq=100;
   /** Test tone ampliture (from 0.0 to 1.0) */
   public static double tone_amp=1.0;

   /** Runtime media process */
   Process media_process=null;

   int dir; // duplex= 0, recv-only= -1, send-only= +1;

   SipdroidSocket socket=null;
   RtpStreamSender sender=null;
   RtpStreamReceiver receiver=null;

   //change DTMF
   boolean useDTMF = false;  // zero means not use outband DTMF

   /** Costructs the audio launcher */
   public JAudioLauncher(RtpStreamSender rtp_sender, RtpStreamReceiver rtp_receiver, Log logger)
   {  log=logger;
      sender=rtp_sender;
      receiver=rtp_receiver;
   }

   /** Costructs the audio launcher */
   public JAudioLauncher(int local_port, String remote_addr, int remote_port, int direction, String audiofile_in, String
audiofile_out, int sample_rate, int sample_size, int frame_size, Log logger, Codecs.Map payload_type, int dtmf_pt)
   {  log=logger;
      frame_rate=sample_rate/frame_size;
      useDTMF = (dtmf_pt != 0);
      try
      {
            CallRecorder call_recorder = null;
            if
(PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).getBoolean(org.sipdroid.sipua.ui.Settings.PREF_CALLREC
ORD,
                                                      org.sipdroid.sipua.ui.Settings.DEFAULT_CALLRECORD))
                        call_recorder = new CallRecorder(null,payload_type.codec.samp_rate()); // Autogenerate filename
from date.
            socket=new SipdroidSocket(local_port);
         dir=direction;
         // sender
         if (dir>=0)
         {  printLog("new audio sender to "+remote_addr+":"+remote_port,LogLevel.MEDIUM);
            //audio_input=new AudioInput();
```

```
            sender=new
RtpStreamSender(true,payload_type,frame_rate,frame_size,socket,remote_addr,remote_port,call_recorder);
            sender.setSyncAdj(2);
            sender.setDTMFpayloadType(dtmf_pt);
         }

         // receiver
         if (dir<=0)
         {  printLog("new audio receiver on "+local_port,LogLevel.MEDIUM);
            receiver=new RtpStreamReceiver(socket,payload_type,call_recorder);
         }
      }
      catch (Exception e) {  printException(e,LogLevel.HIGH);  }
   }

   /** Starts media application */
   public boolean startMedia()
   {  printLog("starting java audio..",LogLevel.HIGH);

      if (sender!=null)
      {  printLog("start sending",LogLevel.LOW);
         sender.start();
      }
      if (receiver!=null)
      {  printLog("start receiving",LogLevel.LOW);
         receiver.start();
      }

      return true;
   }

   /** Stops media application */
   public boolean stopMedia()
   {  printLog("halting java audio..",LogLevel.HIGH);
      if (sender!=null)
      {  sender.halt(); sender=null;
         printLog("sender halted",LogLevel.LOW);
      }
      if (receiver!=null)
      {  receiver.halt(); receiver=null;
         printLog("receiver halted",LogLevel.LOW);
      }
      if (socket != null)
            socket.close();
      return true;
   }

   public boolean muteMedia()
   {
             if (sender != null)
                       return sender.mute();
             return false;
   }

   public int speakerMedia(int mode)
   {
             if (receiver != null)
                       return receiver.speaker(mode);
             return 0;
   }

   public void bluetoothMedia()
   {
             if (receiver != null)
                       receiver.bluetooth();
   }

   //change DTMF
           /** Send outband DTMF packets **/
  public boolean sendDTMF(char c){
              if (! useDTMF) return false;
              sender.sendDTMF(c);
              return true;
  }

   // ***************************** Logs ****************************

   /** Adds a new string to the default Log */
   private void printLog(String str)
   {  printLog(str,LogLevel.HIGH);
   }

   /** Adds a new string to the default Log */
   private void printLog(String str, int level)
   {
             if (Sipdroid.release) return;
             if (log!=null) log.println("AudioLauncher: "+str,level+SipStack.LOG_LEVEL_UA);
      if (level<=LogLevel.HIGH) System.out.println("AudioLauncher: "+str);
   }

   /** Adds the Exception message to the default Log */
   void printException(Exception e,int level)
   {
             if (Sipdroid.release) return;
             if (log!=null) log.printException(e,level+SipStack.LOG_LEVEL_UA);
```

```
    if (level<=LogLevel.HIGH) e.printStackTrace();
  }
```

MediaLauncher.java

```
package org.sipdroid.media;

/** Interface for classes that start media application such as for audio or video */
public interface MediaLauncher {
          /** Starts media application */
          public boolean startMedia();

          /** Stops media application */
          public boolean stopMedia();

          public boolean muteMedia();
          public int speakerMedia(int mode);
          public void bluetoothMedia();

          public boolean sendDTMF(char c);
                    }
```

➢ /Sipdroid/src/org/sipdroid/net

KeepAliveSip.java

```
package org.sipdroid.net;

import org.zoolu.sip.provider.SipProvider;
import org.zoolu.sip.message.Message;

/**
 * KeepAliveSip thread, for keeping the connection up toward a target SIP node
 * (e.g. toward the serving proxy/gw or a remote UA). It periodically sends
 * keep-alive tokens in order to refresh TCP connection timeouts and/or NAT
 * TCP/UDP session timeouts.
 */
public class KeepAliveSip extends KeepAliveUdp {
    /** SipProvider */
    SipProvider sip_provider;

    /** Sip message */
    Message message = null;

    /** Creates a new SIP KeepAliveSip daemon */
    public KeepAliveSip(SipProvider sip_provider,
                        long delta_time) {
              super(null, delta_time);
              init(sip_provider, null);
              start();
    }

    /** Creates a new SIP KeepAliveSip daemon */
    public KeepAliveSip(SipProvider sip_provider,
                        Message message, long delta_time) {
              super(null, delta_time);
              init(sip_provider, message);
              start();
    }

    /** Inits the KeepAliveSip in SIP mode */
    private void init(SipProvider sip_provider, Message message) {
              this.sip_provider = sip_provider;
              if (message == null) {
                        message = new Message("\r\n");
              }
              this.message = message;
    }

    /** Sends the kepp-alive packet now. */
    public void sendToken() throws java.io.IOException { // do send?
              if (!stop && sip_provider != null) {
                        sip_provider.sendMessage(message);
              }
    }

    /** Main thread. */
    public void run() {
              super.run();
              sip_provider = null;
    }

    /** Gets a String representation of the Object */
    public String toString() {
              String str = null;
              if (sip_provider != null) {
                        str = "sip:" + sip_provider.getViaAddress() + ":"
                                             + sip_provider.getPort();
              }
              return str + " (" + delta_time + "ms)";
    }
```

KeepAliveUDP.java

```
package org.sipdroid.net;
```

```
import org.zoolu.net.*;

/**
 * KeepAliveUdp thread, for keeping the connection up toward a target node (e.g.
 * toward the serving proxy/gw or a remote UA). It periodically sends
 * keep-alive tokens in order to refresh NAT UDP session timeouts.
 * <p>
 * It can be used for both signaling (SIP) or data plane (RTP/UDP).
 */
public class KeepAliveUdp extends Thread {
	/** Destination socket address (e.g. the registrar server) */
	protected SocketAddress target;

	/** Time between two keep-alive tokens [millisecs] */
	protected long delta_time;

	/** UdpSocket */
	UdpSocket udp_socket;

	/** Udp packet */
	UdpPacket udp_packet = null;

	/** Expiration date [millisecs] */
	long expire = 0;

	/** Whether it is running */
	boolean stop = false;

	/** Creates a new KeepAliveUdp daemon */
	protected KeepAliveUdp(SocketAddress target, long delta_time) {
		this.target = target;
		this.delta_time = delta_time;
	}

	/** Creates a new KeepAliveUdp daemon */
	public KeepAliveUdp(UdpSocket udp_socket, SocketAddress target,
				long delta_time) {
		this.target = target;
		this.delta_time = delta_time;
		init(udp_socket, null);
		start();
	}

	/** Creates a new KeepAliveUdp daemon */
	public KeepAliveUdp(UdpSocket udp_socket, SocketAddress target,
					UdpPacket udp_packet, long delta_time) {
		this.target = target;
		this.delta_time = delta_time;
		init(udp_socket, udp_packet);
		start();
	}

	/** Inits the KeepAliveUdp */
	private void init(UdpSocket udp_socket, UdpPacket udp_packet) {
		this.udp_socket = udp_socket;
		if (udp_packet == null) {
			byte[] buff = { (byte) '\r', (byte) '\n' };
			udp_packet = new UdpPacket(buff, 0, buff.length);
		}
		if (target != null) {
			udp_packet.setIpAddress(target.getAddress());
			udp_packet.setPort(target.getPort());
		}
		this.udp_packet = udp_packet;
	}

	/** Whether the UDP relay is running */
	public boolean isRunning() {
		return !stop;
	}

	/** Sets the time (in milliseconds) between two keep-alive tokens */
	public void setDeltaTime(long delta_time) {
		this.delta_time = delta_time;
	}

	/** Gets the time (in milliseconds) between two keep-alive tokens */
	public long getDeltaTime() {
		return delta_time;
	}

	/** Sets the destination SocketAddress */
	public void setDestSoAddress(SocketAddress soaddr) {
		target = soaddr;
		if (udp_packet != null && target != null) {
			udp_packet.setIpAddress(target.getAddress());
			udp_packet.setPort(target.getPort());
		}

	}

	/** Gets the destination SocketAddress */
	public SocketAddress getDestSoAddress() {
		return target;
```

```
	}

	/** Sets the expiration time (in milliseconds) */
	public void setExpirationTime(long time) {
		if (time == 0)
			expire = 0;
		else
			expire = System.currentTimeMillis() + time;
	}

	/** Stops sending keep-alive tokens */
	public void halt() {
		stop = true;
	}

	/** Sends the kepp-alive packet now. */
	public void sendToken() throws java.io.IOException { // do send?
		if (!stop && target != null && udp_socket != null) {
			udp_socket.send(udp_packet);
		}
	}

	/** Main thread. */
	public void run() {
		try {
			while (!stop) {
//				sendToken();
				// System.out.print(".");
				sleep(delta_time);
				if (expire > 0 && System.currentTimeMillis() > expire)
					halt();
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
		// System.out.println("o");
		udp_socket = null;
	}

	/** Gets a String representation of the Object */
	public String toString() {
		String str = null;
		if (udp_socket != null) {
			str = "udp:" + udp_socket.getLocalAddress() + ":"
				+ udp_socket.getLocalPort() + "-->" + target.toString();
		}
		return str + " (" + delta_time + "ms)";
	}
```

RtpPacket.java

```
package org.sipdroid.net;
import org.zoolu.tools.Random;
/**
 * RtpPacket implements a RTP packet.
 */
public class RtpPacket {
	/* RTP packet buffer containing both the RTP header and payload */
	byte[] packet;

	/* RTP packet length */
	int packet_len;

	/* RTP header length */
	// int header_len;
	/** Gets the RTP packet */
	public byte[] getPacket() {
		return packet;
	}

	/** Gets the RTP packet length */
	public int getLength() {
		return packet_len;
	}

	/** Gets the RTP header length */
	public int getHeaderLength() {
		if (packet_len >= 12)
			return 12 + 4 * getCscrCount();
		else
			return packet_len; // broken packet
	}

	/** Gets the RTP header length */
	public int getPayloadLength() {
		if (packet_len >= 12)
			return packet_len - getHeaderLength();
		else
			return 0; // broken packet
	}

	/** Sets the RTP payload length */
	public void setPayloadLength(int len) {
		packet_len = getHeaderLength() + len;
```

```
}

/** Gets the version (V) */
public int getVersion() {
	if (packet_len >= 12)
		return (packet[0] >> 6 & 0x03);
	else
		return 0; // broken packet
}

/** Sets the version (V) */
public void setVersion(int v) {
	if (packet_len >= 12)
		packet[0] = (byte) ((packet[0] & 0x3F) | ((v & 0x03) << 6));
}

/** Whether has padding (P) */
public boolean hasPadding() {
	if (packet_len >= 12)
		return getBit(packet[0], 5);
	else
		return false; // broken packet
}

/** Set padding (P) */
public void setPadding(boolean p) {
	if (packet_len >= 12)
		packet[0] = setBit(p, packet[0], 5);
}

/** Whether has extension (X) */
public boolean hasExtension() {
	if (packet_len >= 12)
		return getBit(packet[0], 4);
	else
		return false; // broken packet
}

/** Set extension (X) */
public void setExtension(boolean x) {
	if (packet_len >= 12)
		packet[0] = setBit(x, packet[0], 4);
}

/** Gets the CSCR count (CC) */
public int getCscrCount() {
	if (packet_len >= 12)
		return (packet[0] & 0x0F);
	else
		return 0; // broken packet
}

/** Whether has marker (M) */
public boolean hasMarker() {
	if (packet_len >= 12)
		return getBit(packet[1], 7);
	else
		return false; // broken packet
}

/** Set marker (M) */
public void setMarker(boolean m) {
	if (packet_len >= 12)
		packet[1] = setBit(m, packet[1], 7);
}

/** Gets the payload type (PT) */
public int getPayloadType() {
	if (packet_len >= 12)
		return (packet[1] & 0x7F);
	else
		return -1; // broken packet
}

/** Sets the payload type (PT) */
public void setPayloadType(int pt) {
	if (packet_len >= 12)
		packet[1] = (byte) ((packet[1] & 0x80) | (pt & 0x7F));
}

/** Gets the sequence number */
public int getSequenceNumber() {
	if (packet_len >= 12)
		return getInt(packet, 2, 4);
	else
		return 0; // broken packet
}

/** Sets the sequence number */
public void setSequenceNumber(int sn) {
	if (packet_len >= 12)
		setInt(sn, packet, 2, 4);
}

/** Gets the timestamp */
```

```
public long getTimestamp() {
		if (packet_len >= 12)
			return getLong(packet, 4, 8);
		else
			return 0; // broken packet
}

/** Sets the timestamp */
public void setTimestamp(long timestamp) {
		if (packet_len >= 12)
			setLong(timestamp, packet, 4, 8);
}

/** Gets the SSCR */
public long getSscr() {
		if (packet_len >= 12)
			return getLong(packet, 8, 12);
		else
			return 0; // broken packet
}

/** Sets the SSCR */
public void setSscr(long ssrc) {
		if (packet_len >= 12)
			setLong(ssrc, packet, 8, 12);
}

/** Gets the CSCR list */
public long[] getCscrList() {
		int cc = getCscrCount();
		long[] cscr = new long[cc];
		for (int i = 0; i < cc; i++)
			cscr[i] = getLong(packet, 12 + 4 * i, 16 + 4 * i);
		return cscr;
}

/** Sets the CSCR list */
public void setCscrList(long[] cscr) {
		if (packet_len >= 12) {
			int cc = cscr.length;
			if (cc > 15)
				cc = 15;
			packet[0] = (byte) (((packet[0] >> 4) << 4) + cc);
			cscr = new long[cc];
			for (int i = 0; i < cc; i++)
				setLong(cscr[i], packet, 12 + 4 * i, 16 + 4 * i);
			// header_len=12+4*cc;
		}
}

/** Sets the payload */
public void setPayload(byte[] payload, int len) {
		if (packet_len >= 12) {
			int header_len = getHeaderLength();
			for (int i = 0; i < len; i++)
				packet[header_len + i] = payload[i];
			packet_len = header_len + len;
		}
}

/** Gets the payload */
public byte[] getPayload() {
		int header_len = getHeaderLength();
		int len = packet_len - header_len;
		byte[] payload = new byte[len];
		for (int i = 0; i < len; i++)
			payload[i] = packet[header_len + i];
		return payload;
}

/** Creates a new RTP packet */
public RtpPacket(byte[] buffer, int packet_length) {
		packet = buffer;
		packet_len = packet_length;
		if (packet_len < 12)
			packet_len = 12;
		init(0x0F);
}

/** init the RTP packet header (only PT) */
public void init(int ptype) {
		init(ptype, Random.nextLong());
}

/** init the RTP packet header (PT and SSCR) */
public void init(int ptype, long sscr) {
		init(ptype, Random.nextInt(), Random.nextLong(), sscr);
}

/** init the RTP packet header (PT, SQN, TimeStamp, SSCR) */
public void init(int ptype, int seqn, long timestamp, long sscr) {
		setVersion(2);
		setPayloadType(ptype);
		setSequenceNumber(seqn);
		setTimestamp(timestamp);
```

```
                setSscr(sscr);
        }

        // ********************** Private and Static ***********************

        /** Gets int value */
        private static int getInt(byte b) {
                return ((int) b + 256) % 256;
        }

        /** Gets long value */
        private static long getLong(byte[] data, int begin, int end) {
                long n = 0;
                for (; begin < end; begin++) {
                        n <<= 8;
                        n += data[begin] & 0xFF;
                }
                return n;
        }

        /** Sets long value */
        private static void setLong(long n, byte[] data, int begin, int end) {
                for (end--; end >= begin; end--) {
                        data[end] = (byte) (n % 256);
                        n >>= 8;
                }
        }

        /** Gets Int value */
        private static int getInt(byte[] data, int begin, int end) {
                return (int) getLong(data, begin, end);
        }

        /** Sets Int value */
        private static void setInt(int n, byte[] data, int begin, int end) {
                setLong(n, data, begin, end);
        }

        /** Gets bit value */
        private static boolean getBit(byte b, int bit) {
                return (b >> bit) == 1;
        }

        /** Sets bit value */
        private static byte setBit(boolean value, byte b, int bit) {
                if (value)
                        return (byte) (b | (1 << bit));
                else
                        return (byte) ((b | (1 << bit)) ^ (1 << bit));
        }
}
```

RtpSocket.java

```
package org.sipdroid.net;
import java.net.InetAddress;
import java.net.DatagramPacket;
import java.io.IOException;

/**
 * RtpSocket implements a RTP socket for receiving and sending RTP packets.
 * <p>
 * RtpSocket is associated to a DatagramSocket that is used to send and/or
 * receive RtpPackets.
 */
public class RtpSocket {
        /** UDP socket */
        SipdroidSocket socket;
        DatagramPacket datagram;

        /** Remote address */
        InetAddress r_addr;

        /** Remote port */
        int r_port;

        /** Creates a new RTP socket (only receiver) */
        public RtpSocket(SipdroidSocket datagram_socket) {
                socket = datagram_socket;
                r_addr = null;
                r_port = 0;
                datagram = new DatagramPacket(new byte[1],1);
        }

        /** Creates a new RTP socket (sender and receiver) */
        public RtpSocket(SipdroidSocket datagram_socket,
                        InetAddress remote_address, int remote_port) {
                socket = datagram_socket;
                r_addr = remote_address;
                r_port = remote_port;
                datagram = new DatagramPacket(new byte[1],1);
        }
        /** Returns the RTP SipdroidSocket */
        public SipdroidSocket getDatagramSocket() {
                return socket;
```

```
	}

	/** Receives a RTP packet from this socket */
	public void receive(RtpPacket rtpp) throws IOException {
		datagram.setData(rtpp.packet);
		datagram.setLength(rtpp.packet.length);
		socket.receive(datagram);
		if (!socket.isConnected())
			socket.connect(datagram.getAddress(),datagram.getPort());
		rtpp.packet_len = datagram.getLength();
	}

	/** Sends a RTP packet from this socket */
	public void send(RtpPacket rtpp) throws IOException {
		datagram.setData(rtpp.packet);
		datagram.setLength(rtpp.packet_len);
		datagram.setAddress(r_addr);
		datagram.setPort(r_port);
		socket.send(datagram);
	}
	/** Closes this socket */
	public void close() { // socket.close();
	}

}
```

SipdroidSocket.java

```
package org.sipdroid.net;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.SocketOptions;
import java.net.UnknownHostException;

import org.sipdroid.net.impl.OSNetworkSystem;
import org.sipdroid.net.impl.PlainDatagramSocketImpl;

public class SipdroidSocket extends DatagramSocket {

	PlainDatagramSocketImpl impl;
	public static boolean loaded = false;

	public SipdroidSocket(int port) throws SocketException, UnknownHostException {
		super(!loaded?port:0);
		if (loaded) {
			impl = new PlainDatagramSocketImpl();
			impl.create();
			impl.bind(port,InetAddress.getByName("0"));
		}
	}

	public void close() {
		super.close();
		if (loaded) impl.close();
	}

	public void setSoTimeout(int val) throws SocketException {
		if (loaded) impl.setOption(SocketOptions.SO_TIMEOUT, val);
		else super.setSoTimeout(val);
	}

	public void receive(DatagramPacket pack) throws IOException {
		if (loaded) impl.receive(pack);
		else super.receive(pack);
	}

	public void send(DatagramPacket pack) throws IOException {
		if (loaded) impl.send(pack);
		else super.send(pack);
	}

	public boolean isConnected() {
		if (loaded) return true;
		else return super.isConnected();
	}

	public void disconnect() {
		if (!loaded) super.disconnect();
	}

	public void connect(InetAddress addr,int port) {
		if (!loaded) super.connect(addr,port);
	}

	static {
			   try {
			System.loadLibrary("OSNetworkSystem");
			OSNetworkSystem.getOSNetworkSystem().oneTimeInitialization(true);
			SipdroidSocket.loaded = true;
			   } catch (Throwable e) {
			   }
	}
```

➢ /Sipdroid/src/org/sipdroid/sipua

UserAgent.java

```
package org.sipdroid.sipua;
import java.util.Enumeration;
import java.util.Vector;
import org.sipdroid.codecs.Codec;
import org.sipdroid.codecs.Codecs;
import org.sipdroid.media.JAudioLauncher;
import org.sipdroid.media.MediaLauncher;
import org.sipdroid.media.RtpStreamReceiver;
import org.sipdroid.sipua.ui.Receiver;
import org.sipdroid.sipua.ui.Settings;
import org.sipdroid.sipua.ui.Sipdroid;
import org.zoolu.net.IpAddress;
import org.zoolu.sdp.AttributeField;
import org.zoolu.sdp.ConnectionField;
import org.zoolu.sdp.MediaDescriptor;
import org.zoolu.sdp.MediaField;
import org.zoolu.sdp.SessionDescriptor;
import org.zoolu.sdp.TimeField;
import org.zoolu.sip.address.NameAddress;
import org.zoolu.sip.call.Call;
import org.zoolu.sip.call.CallListenerAdapter;
import org.zoolu.sip.call.ExtendedCall;
import org.zoolu.sip.call.SdpTools;
import org.zoolu.sip.header.StatusLine;
import org.zoolu.sip.message.Message;
import org.zoolu.sip.provider.SipProvider;
import org.zoolu.sip.provider.SipStack;
import org.zoolu.tools.Log;
import org.zoolu.tools.LogLevel;
import org.zoolu.tools.Parser;

/**
 * Simple SIP user agent (UA). It includes audio/video applications.
 * <p>
 * It can use external audio/video tools as media applications. Currently only
 * RAT (Robust Audio Tool) and VIC are supported as external applications.
 */
public class UserAgent extends CallListenerAdapter {
           /** Event logger. */
           Log log;

           /** UserAgentProfile */
           public UserAgentProfile user_profile;

           /** SipProvider */
           protected SipProvider sip_provider;

           /** Call */
           // Call call;
           protected ExtendedCall call;

           /** Call transfer */
           protected ExtendedCall call_transfer;

           /** Audio application */
           public MediaLauncher audio_app = null;

           /** Local sdp */
           protected String local_session = null;

           public static final int UA_STATE_IDLE = 0;
           public static final int UA_STATE_INCOMING_CALL = 1;
           public static final int UA_STATE_OUTGOING_CALL = 2;
           public static final int UA_STATE_INCALL = 3;
           public static final int UA_STATE_HOLD = 4;

           int call_state = UA_STATE_IDLE;
           String remote_media_address;
           int remote_video_port,local_video_port;

           // ************************** Basic methods **************************

           /** Changes the call state */
           protected synchronized void changeStatus(int state,String caller) {
                      call_state = state;
                      Receiver.onState(state, caller);
           }

           protected void changeStatus(int state) {
                      changeStatus(state, null);
           }

           /** Checks the call state */
           protected boolean statusIs(int state) {
                      return (call_state == state);
           }

           /**
            * Sets the automatic answer time (default is -1 that means no auto accept
            * mode)
            */
```

```
	public void setAcceptTime(int accept_time) {
		user_profile.accept_time = accept_time;
	}

	/**
	 * Sets the automatic hangup time (default is 0, that corresponds to manual
	 * hangup mode)
	 */
	public void setHangupTime(int time) {
		user_profile.hangup_time = time;
	}

	/** Sets the redirection url (default is null, that is no redircetion) */
	public void setRedirection(String url) {
		user_profile.redirect_to = url;
	}

	/** Sets the no offer mode for the invite (default is false) */
	public void setNoOfferMode(boolean nooffer) {
		user_profile.no_offer = nooffer;
	}

	/** Enables audio */
	public void setAudio(boolean enable) {
		user_profile.audio = enable;
	}

	/** Sets the receive only mode */
	public void setReceiveOnlyMode(boolean r_only) {
		user_profile.recv_only = r_only;
	}

	/** Sets the send only mode */
	public void setSendOnlyMode(boolean s_only) {
		user_profile.send_only = s_only;
	}

	/** Sets the send tone mode */
	public void setSendToneMode(boolean s_tone) {
		user_profile.send_tone = s_tone;
	}

	/** Sets the send file */

	public void setSendFile(String file_name) {
		user_profile.send_file = file_name;
	}

	/** Sets the recv file */

	public void setRecvFile(String file_name) {
		user_profile.recv_file = file_name;
	}

	/** Gets the local SDP */
	public String getSessionDescriptor() {
		return local_session;
	}

	//change start (multi codecs)
	/** Inits the local SDP (no media spec) */
	public void initSessionDescriptor(Codecs.Map c) {
		SessionDescriptor sdp = new SessionDescriptor(
					user_profile.from_url,
					sip_provider.getViaAddress());

		local_session = sdp.toString();

		//We will have at least one media line, and it will be
		//audio
		if (user_profile.audio || !user_profile.video)
		{
//			addMediaDescriptor("audio", user_profile.audio_port, c,
user_profile.audio_sample_rate);
			addMediaDescriptor("audio", user_profile.audio_port, c);
		}

		if (user_profile.video)
		{
			addMediaDescriptor("video", user_profile.video_port,
						user_profile.video_avp, "h263-1998", 90000);
		}
	}
	//change end

	/** Adds a single media to the SDP */
	private void addMediaDescriptor(String media, int port, int avp,
						String codec, int rate) {
		SessionDescriptor sdp = new SessionDescriptor(local_session);

		String attr_param = String.valueOf(avp);

		if (codec != null)
		{
			attr_param += " " + codec + "/" + rate;
```

```
                    }
                    sdp.addMedia(new MediaField(media, port, 0, "RTP/AVP",
                                        String.valueOf(avp)),
                                        new AttributeField("rtpmap", attr_param));

                    local_session = sdp.toString();
          }

          /** Adds a set of media to the SDP */
//        private void addMediaDescriptor(String media, int port, Codecs.Map c,int rate) {
          private void addMediaDescriptor(String media, int port, Codecs.Map c) {
                    SessionDescriptor sdp = new SessionDescriptor(local_session);

                    Vector<String> avpvec = new Vector<String>();
                    Vector<AttributeField> afvec = new Vector<AttributeField>();
                    if (c == null) {
                              // offer all known codecs
                              for (int i : Codecs.getCodecs()) {
                                        Codec codec = Codecs.get(i);
                                        if (i == 0) codec.init();
                                        avpvec.add(String.valueOf(i));
                                        if (codec.number() == 9)
                                                  afvec.add(new AttributeField("rtpmap", String.format("%d %s/%d",
i, codec.userName(), 8000))); // kludge for G722. See RFC3551.
                                        else
                                                  afvec.add(new AttributeField("rtpmap", String.format("%d %s/%d",
i, codec.userName(), codec.samp_rate())));
                              }
                    } else {
                              c.codec.init();
                              avpvec.add(String.valueOf(c.number));
                              if (c.codec.number() == 9)
                                        afvec.add(new AttributeField("rtpmap", String.format("%d %s/%d", c.number,
c.codec.userName(), 8000))); // kludge for G722. See RFC3551.
                              else
                                        afvec.add(new AttributeField("rtpmap", String.format("%d %s/%d", c.number,
c.codec.userName(), c.codec.samp_rate())));
                    }
                    if (user_profile.dtmf_avp != 0){
                              avpvec.add(String.valueOf(user_profile.dtmf_avp));
                              afvec.add(new AttributeField("rtpmap", String.format("%d telephone-event/%d",
user_profile.dtmf_avp, user_profile.audio_sample_rate)));
                              afvec.add(new AttributeField("fmtp", String.format("%d 0-15", user_profile.dtmf_avp)));
                    }

                    //String attr_param = String.valueOf(avp);

                    sdp.addMedia(new MediaField(media, port, 0, "RTP/AVP", avpvec), afvec);

                    local_session = sdp.toString();
          }

          // ************************** Public Methods *************************

          /** Costructs a UA with a default media port */
          public UserAgent(SipProvider sip_provider, UserAgentProfile user_profile) {
                    this.sip_provider = sip_provider;
                    log = sip_provider.getLog();
                    this.user_profile = user_profile;
                    realm = user_profile.realm;

                    // if no contact_url and/or from_url has been set, create it now
                    user_profile.initContactAddress(sip_provider);
          }

          String realm;

          /** Makes a new call (acting as UAC). */
          public boolean call(String target_url, boolean send_anonymous) {

                    if (Receiver.call_state != UA_STATE_IDLE)
                    {
                              //We can initiate or terminate a call only when
                              //we are in an idle state
                              printLog("Call attempted in state" + this.getSessionDescriptor() + " : Failing
Request", LogLevel.HIGH);
                              return false;
                    }
                    hangup(); // modified
                    changeStatus(UA_STATE_OUTGOING_CALL,target_url);

                    String from_url;

                    if (!send_anonymous)
                    {
                              from_url = user_profile.from_url;
                    }
                    else
                    {
                              from_url = "sip:anonymous@anonymous.com";
                    }

                    //change start multi codecs
                    createOffer();
                    //change end
```

```
                    call = new ExtendedCall(sip_provider, from_url,
                                        user_profile.contact_url, user_profile.username,
                                        user_profile.realm, user_profile.passwd, this);

                    // in case of incomplete url (e.g. only 'user' is present), try to
                    // complete it
                    if (target_url.indexOf("@") < 0) {
                              if (user_profile.realm.equals(Settings.DEFAULT_SERVER))
                                        target_url = "&" + target_url;
                              target_url = target_url + "@" + realm; // modified
                    }

                    // MMTel addition to define MMTel ICSI to be included in INVITE (added by mandrajg)
                    String icsi = null;
                    if (user_profile.mmtel == true){
                              icsi = "\"urn%3Aurn-7%3A3gpp-service.ims.icsi.mmtel\"";
                    }

                    target_url = sip_provider.completeNameAddress(target_url).toString();

                    if (user_profile.no_offer)
                    {
                              call.call(target_url);
                    }
                    else
                    {
                              call.call(target_url, local_session, icsi);           // modified by mandrajg
                    }

                    return true;
          }

          public void info(char c, int duration)
          {
                    boolean use2833 = audio_app != null && audio_app.sendDTMF(c); // send out-band DTMF (rfc2833) if
supported

                    if (!use2833 && call != null)
                              call.info(c, duration);
          }

          /** Waits for an incoming call (acting as UAS). */
          public boolean listen() {

                    if (Receiver.call_state != UA_STATE_IDLE)
                    {
                              //We can listen for a call only when
                              //we are in an idle state
                              printLog("Call listening mode initiated in " + this.getSessionDescriptor() + " :
Failing Request", LogLevel.HIGH);
                              return false;
                    }

                    hangup();

                    call = new ExtendedCall(sip_provider, user_profile.from_url,
                                        user_profile.contact_url, user_profile.username,
                                        user_profile.realm, user_profile.passwd, this);
                    call.listen();

                    return true;
          }

          /** Closes an ongoing, incoming, or pending call */
          public void hangup()
          {
                    printLog("HANGUP");
                    closeMediaApplication();

                    if (call != null)
                    {
                              call.hangup();
                    }

                    changeStatus(UA_STATE_IDLE);
          }

          /** Accepts an incoming call */
          public boolean accept()
          {
                    if (call == null)
                    {
                              return false;
                    }

                    printLog("ACCEPT");
                    changeStatus(UA_STATE_INCALL); // modified

                    call.accept(local_session);

                    return true;
          }

          /** Redirects an incoming call */
          public void redirect(String redirection)
```

```
	{
		if (call != null)
		{
			call.redirect(redirection);
		}
	}

	/** Launches the Media Application (currently, the RAT audio tool) */
	protected void launchMediaApplication() {
		// exit if the Media Application is already running
		if (audio_app != null) {
			printLog("DEBUG: media application is already running",
					LogLevel.HIGH);
			return;
		}
		Codecs.Map c;
		// parse local sdp
		SessionDescriptor local_sdp = new SessionDescriptor(call
				.getLocalSessionDescriptor());
		int local_audio_port = 0;
		local_video_port = 0;
		int dtmf_pt = 0;
		c = Codecs.getCodec(local_sdp);
		if (c == null) {
			Receiver.call_end_reason = R.string.card_title_ended_no_codec;
			hangup();
			return;
		}
		MediaDescriptor m = local_sdp.getMediaDescriptor("video");
		if ( m != null)
			local_video_port = m.getMedia().getPort();
		m = local_sdp.getMediaDescriptor("audio");
		if (m != null) {
			local_audio_port = m.getMedia().getPort();
			if (m.getMedia().getFormatList().contains(String.valueOf(user_profile.dtmf_avp)))
				dtmf_pt = user_profile.dtmf_avp;
		}
		// parse remote sdp
		SessionDescriptor remote_sdp = new SessionDescriptor(call
				.getRemoteSessionDescriptor());
		remote_media_address = (new Parser(remote_sdp.getConnection()
				.toString())).skipString().skipString().getString();
		int remote_audio_port = 0;
		remote_video_port = 0;
		for (Enumeration<MediaDescriptor> e = remote_sdp.getMediaDescriptors()
				.elements(); e.hasMoreElements();) {
			MediaField media = e.nextElement().getMedia();
			if (media.getMedia().equals("audio"))
				remote_audio_port = media.getPort();
			if (media.getMedia().equals("video"))
				remote_video_port = media.getPort();
		}

		// select the media direction (send_only, recv_ony, fullduplex)
		int dir = 0;
		if (user_profile.recv_only)
			dir = -1;
		else if (user_profile.send_only)
			dir = 1;

		if (user_profile.audio && local_audio_port != 0
				&& remote_audio_port != 0) { // create an audio_app and start

	// it

			if (audio_app == null) { // for testing..
				String audio_in = null;
				if (user_profile.send_tone) {
					audio_in = JAudioLauncher.TONE;
				} else if (user_profile.send_file != null) {
					audio_in = user_profile.send_file;
				}
				String audio_out = null;
				if (user_profile.recv_file != null) {
					audio_out = user_profile.recv_file;
				}

				audio_app = new JAudioLauncher(local_audio_port,
						remote_media_address, remote_audio_port, dir,
audio_in,
						audio_out, c.codec.samp_rate(),
						user_profile.audio_sample_size,
						c.codec.frame_size(), log, c, dtmf_pt);
			}
			audio_app.startMedia();
		}
	}

	/** Close the Media Application */
	protected void closeMediaApplication() {
		if (audio_app != null) {
			audio_app.stopMedia();
			audio_app = null;
		}
	}
```

```
public boolean muteMediaApplication() {
          if (audio_app != null)
                    return audio_app.muteMedia();
          return false;
}

public int speakerMediaApplication(int mode) {
          int old;

          if (audio_app != null)
                    return audio_app.speakerMedia(mode);
          old = RtpStreamReceiver.speakermode;
          RtpStreamReceiver.speakermode = mode;
          return old;
}

public void bluetoothMediaApplication() {
          if (audio_app != null)
                    audio_app.bluetoothMedia();
}

private void createOffer() {
          initSessionDescriptor(null);
}

private void createAnswer(SessionDescriptor remote_sdp) {

          Codecs.Map c = Codecs.getCodec(remote_sdp);
          if (c == null)
                    throw new RuntimeException("Failed to get CODEC: AVAILABLE : " + remote_sdp);
          initSessionDescriptor(c);
          sessionProduct(remote_sdp);
}

private void sessionProduct(SessionDescriptor remote_sdp) {
          SessionDescriptor local_sdp = new SessionDescriptor(local_session);
          SessionDescriptor new_sdp = new SessionDescriptor(local_sdp
                              .getOrigin(), local_sdp.getSessionName(), local_sdp
                              .getConnection(), local_sdp.getTime());
          new_sdp.addMediaDescriptors(local_sdp.getMediaDescriptors());
          new_sdp = SdpTools.sdpMediaProduct(new_sdp, remote_sdp
                              .getMediaDescriptors());
          //new_sdp = SdpTools.sdpAttirbuteSelection(new_sdp, "rtpmap"); ////change multi codecs
          local_session = new_sdp.toString();
          if (call!=null) call.setLocalSessionDescriptor(local_session);
}

// ********************** Call callback functions **********************

/**
 * Callback function called when arriving a new INVITE method (incoming
 * call)
 */
public void onCallIncoming(Call call, NameAddress callee,
                    NameAddress caller, String sdp, Message invite) {
          printLog("onCallIncoming()", LogLevel.LOW);

          if (call != this.call) {
                    printLog("NOT the current call", LogLevel.LOW);
                    return;
          }
          printLog("INCOMING", LogLevel.HIGH);
          int i = 0;
          for (UserAgent ua : Receiver.mSipdroidEngine.uas) {
                    if (ua == this) break;
                    i++;
          }
          if (Receiver.call_state != UA_STATE_IDLE || !Receiver.isFast(i)) {
                    call.busy();
                    listen();
                    return;
          }

          if (Receiver.mSipdroidEngine != null)
                    Receiver.mSipdroidEngine.ua = this;
          changeStatus(UA_STATE_INCOMING_CALL,caller.toString());

          if (sdp == null) {
                    createOffer();
          }
          else {
                    SessionDescriptor remote_sdp = new SessionDescriptor(sdp);
                    try {
                              createAnswer(remote_sdp);
                    } catch (Exception e) {
                              // only known exception is no codec
                              Receiver.call_end_reason = R.string.card_title_ended_no_codec;
                              changeStatus(UA_STATE_IDLE);
                              return;
                    }
          }
          call.ring(local_session);
          launchMediaApplication();
}
```

```
/**
 * Callback function called when arriving a new Re-INVITE method
 * (re-inviting/call modify)
 */
public void onCallModifying(Call call, String sdp, Message invite)
{
          printLog("onCallModifying()", LogLevel.LOW);
          if (call != this.call)
          {
                    printLog("NOT the current call", LogLevel.LOW);
                    return;
          }
          printLog("RE-INVITE/MODIFY", LogLevel.HIGH);

          // to be implemented.
          // currently it simply accepts the session changes (see method
          // onCallModifying() in CallListenerAdapter)
          super.onCallModifying(call, sdp, invite);
}

/**
 * Callback function that may be overloaded (extended). Called when arriving
 * a 180 Ringing or a 183 Session progress with SDP
 */
public void onCallRinging(Call call, Message resp) {
          printLog("onCallRinging()", LogLevel.LOW);
          if (call != this.call && call != call_transfer)
          {
                    printLog("NOT the current call", LogLevel.LOW);
                    return;
          }

          String remote_sdp = call.getRemoteSessionDescriptor();
          if (remote_sdp==null || remote_sdp.length()==0) {
                    printLog("RINGING", LogLevel.HIGH);
                    RtpStreamReceiver.ringback(true);
          }
          else {
                    printLog("RINGING(with SDP)", LogLevel.HIGH);
                    if (! user_profile.no_offer) {
                              RtpStreamReceiver.ringback(false);
                              // Update the local SDP along with offer/answer
                              sessionProduct(new SessionDescriptor(remote_sdp));
                              launchMediaApplication();
                    }
          }
}

/** Callback function called when arriving a 2xx (call accepted) */
public void onCallAccepted(Call call, String sdp, Message resp)
{
          printLog("onCallAccepted()", LogLevel.LOW);

          if (call != this.call && call != call_transfer) {
                    printLog("NOT the current call", LogLevel.LOW);
                    return;
          }

          printLog("ACCEPTED/CALL", LogLevel.HIGH);

          if (!statusIs(UA_STATE_OUTGOING_CALL)) { // modified
                    hangup();
                    return;
          }
          changeStatus(UA_STATE_INCALL);

          SessionDescriptor remote_sdp = new SessionDescriptor(sdp);
          if (user_profile.no_offer) {
                    // answer with the local sdp
                    createAnswer(remote_sdp);
                    call.ackWithAnswer(local_session);
          } else {
                    // Update the local SDP along with offer/answer
                    sessionProduct(remote_sdp);
          }
          launchMediaApplication();

          if (call == call_transfer)
          {
                    StatusLine status_line = resp.getStatusLine();
                    int code = status_line.getCode();
                    String reason = status_line.getReason();
                    this.call.notify(code, reason);
          }
}

/** Callback function called when arriving an ACK method (call confirmed) */
public void onCallConfirmed(Call call, String sdp, Message ack)
{
          printLog("onCallConfirmed()", LogLevel.LOW);

          if (call != this.call) {
                    printLog("NOT the current call", LogLevel.LOW);
                    return;
```

```
                }

                printLog("CONFIRMED/CALL", LogLevel.HIGH);

//              changeStatus(UA_STATE_INCALL); modified

                if (user_profile.hangup_time > 0)
                {
                        this.automaticHangup(user_profile.hangup_time);
                }
        }

        /** Callback function called when arriving a 2xx (re-invite/modify accepted) */
        public void onCallReInviteAccepted(Call call, String sdp, Message resp) {
                printLog("onCallReInviteAccepted()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("RE-INVITE-ACCEPTED/CALL", LogLevel.HIGH);
                if (statusIs(UA_STATE_HOLD))
                        changeStatus(UA_STATE_INCALL);
                else
                        changeStatus(UA_STATE_HOLD);
        }

        /** Callback function called when arriving a 4xx (re-invite/modify failure) */
        public void onCallReInviteRefused(Call call, String reason, Message resp) {
                printLog("onCallReInviteRefused()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("RE-INVITE-REFUSED (" + reason + ")/CALL", LogLevel.HIGH);
        }

        /** Callback function called when arriving a 4xx (call failure) */
        public void onCallRefused(Call call, String reason, Message resp) {
                printLog("onCallRefused()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("REFUSED (" + reason + ")", LogLevel.HIGH);
                if (reason.equalsIgnoreCase("not acceptable here")) {
                        // bummer we have to string compare, this is sdp 488
                        Receiver.call_end_reason = R.string.card_title_ended_no_codec;
                }
                changeStatus(UA_STATE_IDLE);

                if (call == call_transfer)
                {
                        StatusLine status_line = resp.getStatusLine();
                        int code = status_line.getCode();
                        // String reason=status_line.getReason();
                        this.call.notify(code, reason);
                        call_transfer = null;
                }
        }

        /** Callback function called when arriving a 3xx (call redirection) */
        public void onCallRedirection(Call call, String reason,
                        Vector<String> contact_list, Message resp) {
                printLog("onCallRedirection()", LogLevel.LOW);
                if (call != this.call)
                {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("REDIRECTION (" + reason + ")", LogLevel.HIGH);
                call.call(((String) contact_list.elementAt(0)));
        }

        /**
         * Callback function that may be overloaded (extended). Called when arriving
         * a CANCEL request
         */
        public void onCallCanceling(Call call, Message cancel) {
                printLog("onCallCanceling()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("CANCEL", LogLevel.HIGH);
                changeStatus(UA_STATE_IDLE);
        }

        /** Callback function called when arriving a BYE request */
        public void onCallClosing(Call call, Message bye) {
                printLog("onCallClosing()", LogLevel.LOW);
                if (call != this.call && call != call_transfer) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
```

```
                if (call != call_transfer && call_transfer != null) {
                        printLog("CLOSE PREVIOUS CALL", LogLevel.HIGH);
                        this.call = call_transfer;
                        call_transfer = null;
                        return;
                }
                // else
                printLog("CLOSE", LogLevel.HIGH);
                closeMediaApplication();
                changeStatus(UA_STATE_IDLE);
        }

        /**
         * Callback function called when arriving a response after a BYE request
         * (call closed)
         */
        public void onCallClosed(Call call, Message resp) {
                printLog("onCallClosed()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("CLOSE/OK", LogLevel.HIGH);

                changeStatus(UA_STATE_IDLE);
        }

        /** Callback function called when the invite expires */
        public void onCallTimeout(Call call) {
                printLog("onCallTimeout()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("NOT FOUND/TIMEOUT", LogLevel.HIGH);
                changeStatus(UA_STATE_IDLE);
                if (call == call_transfer) {
                        int code = 408;
                        String reason = "Request Timeout";
                        this.call.notify(code, reason);
                        call_transfer = null;
                }
        }

        // ****************** ExtendedCall callback functions ******************

        /**
         * Callback function called when arriving a new REFER method (transfer
         * request)
         */
        public void onCallTransfer(ExtendedCall call, NameAddress refer_to,
                        NameAddress refered_by, Message refer) {
                printLog("onCallTransfer()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("Transfer to " + refer_to.toString(), LogLevel.HIGH);
                call.acceptTransfer();
                call_transfer = new ExtendedCall(sip_provider, user_profile.from_url,
                                user_profile.contact_url, this);
                call_transfer.call(refer_to.toString(), local_session, null);               // modified by
mandrajg
        }

        /** Callback function called when a call transfer is accepted. */
        public void onCallTransferAccepted(ExtendedCall call, Message resp) {
                printLog("onCallTransferAccepted()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("Transfer accepted", LogLevel.HIGH);
        }

        /** Callback function called when a call transfer is refused. */
        public void onCallTransferRefused(ExtendedCall call, String reason,
                        Message resp) {
                printLog("onCallTransferRefused()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("Transfer refused", LogLevel.HIGH);
        }

        /** Callback function called when a call transfer is successfully completed */
        public void onCallTransferSuccess(ExtendedCall call, Message notify) {
                printLog("onCallTransferSuccess()", LogLevel.LOW);
                if (call != this.call) {
                        printLog("NOT the current call", LogLevel.LOW);
                        return;
                }
                printLog("Transfer successed", LogLevel.HIGH);
                call.hangup();
```

```
	}

	/**
	 * Callback function called when a call transfer is NOT sucessfully
	 * completed
	 */
	public void onCallTransferFailure(ExtendedCall call, String reason,
			Message notify) {
		printLog("onCallTransferFailure()", LogLevel.LOW);
		if (call != this.call) {
			printLog("NOT the current call", LogLevel.LOW);
			return;
		}
		printLog("Transfer failed", LogLevel.HIGH);
	}

	// ************************ Schedule events **********************

	/** Schedules a re-inviting event after <i>delay_time</i> secs. */
	void reInvite(final String contact_url, final int delay_time) {
		SessionDescriptor sdp = new SessionDescriptor(local_session);
		sdp.IncrementOLine();
		final SessionDescriptor new_sdp;
		if (statusIs(UserAgent.UA_STATE_INCALL)) { // modified
			new_sdp = new SessionDescriptor(
					sdp.getOrigin(), sdp.getSessionName(), new ConnectionField(
							"IP4", "0.0.0.0"), new TimeField());
		} else {
			new_sdp = new SessionDescriptor(
					sdp.getOrigin(), sdp.getSessionName(), new ConnectionField(
							"IP4", IpAddress.localIpAddress), new
TimeField());
		}
		new_sdp.addMediaDescriptors(sdp.getMediaDescriptors());
		local_session = sdp.toString();
		(new Thread() {
			public void run() {
				runReInvite(contact_url, new_sdp.toString(), delay_time);
			}
		}).start();

	/** Re-invite. */
	private void runReInvite(String contact, String body, int delay_time) {
		try {
			if (delay_time > 0)
				Thread.sleep(delay_time * 1000);
		} catch (Exception e) {
			e.printStackTrace();
		}
			printLog("RE-INVITING/MODIFYING");
			if (call != null && call.isOnCall()) {
				printLog("REFER/TRANSFER");
				call.modify(contact, body);
			}
	} /** Schedules a call-transfer event after <i>delay_time</i> secs. */
	void callTransfer(final String transfer_to, final int delay_time) {
		// in case of incomplete url (e.g. only 'user' is present), try to
		// complete it
		final String target_url;
		if (transfer_to.indexOf("@") < 0)
			target_url = transfer_to + "@" + realm; // modified
		else
			target_url = transfer_to;
		(new Thread() {
			public void run() {
				runCallTransfer(target_url, delay_time);
			}
		}).start();
	} /** Call-transfer. */
	private void runCallTransfer(String transfer_to, int delay_time) {
		try {
			if (delay_time > 0)
				Thread.sleep(delay_time * 1000);
		} catch (Exception e) {
			e.printStackTrace();
		}
			if (call != null && call.isOnCall()) {
				printLog("REFER/TRANSFER");
				call.transfer(transfer_to);
			}
	}	/** Schedules an automatic answer event after <i>delay_time</i> secs. */
	void automaticAccept(final int delay_time) {
		(new Thread() {
			public void run() {
				runAutomaticAccept(delay_time);
			}
		}).start();
	} /** Automatic answer. */
	private void runAutomaticAccept(int delay_time) {
		try {
			if (delay_time > 0)
				Thread.sleep(delay_time * 1000);
		} catch (Exception e) {
			e.printStackTrace();
		}
```

```
                        if (call != null) {
                                printLog("AUTOMATIC-ANSWER");
                                accept();
                        }
        }

        /** Schedules an automatic hangup event after <i>delay_time</i> secs. */
        void automaticHangup(final int delay_time) {
                (new Thread() {
                        public void run() {
                                runAutomaticHangup(delay_time);
                        }
                }).start();
        }

        /** Automatic hangup. */
        private void runAutomaticHangup(int delay_time) {
                try {
                        if (delay_time > 0)
                                Thread.sleep(delay_time * 1000);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                        if (call != null && call.isOnCall()) {
                                printLog("AUTOMATIC-HANGUP");
                                hangup();
                        }

        }       // ***************************** Logs *****************************

        /** Adds a new string to the default Log */
        void printLog(String str) {
                printLog(str, LogLevel.HIGH);
        }

        /** Adds a new string to the default Log */
        void printLog(String str, int level) {
                if (Sipdroid.release) return;
                if (log != null)
                        log.println("UA: " + str, level + SipStack.LOG_LEVEL_UA);
                if ((user_profile == null || !user_profile.no_prompt)
                                && level <= LogLevel.HIGH)
                        System.out.println("UA: " + str);
        }

        /** Adds the Exception message to the default Log */
        void printException(Exception e, int level) {
                if (Sipdroid.release) return;
                if (log != null)
                        log.printException(e, level + SipStack.LOG_LEVEL_UA);
        }
}
```

UserAgentProfile.java

```
package org.sipdroid.sipua;
import org.zoolu.sip.address.*;
import org.zoolu.sip.provider.SipStack;
import org.zoolu.sip.provider.SipProvider;
import org.zoolu.tools.Configure;
import org.zoolu.tools.Parser;

/**
 * UserProfile maintains the user configuration
 */
public class UserAgentProfile extends Configure {
        /** The default configuration file */

        // ********************* user configurations *********************
        /**
         * User's AOR (Address Of Record), used also as From URL. <p/> The AOR is
         * the SIP address used to register with the user's registrar server (if
         * requested). <br/> The address of the registrar is taken from the hostport
         * field of the AOR, i.e. the value(s) host[:port] after the '@' character.
         * <p/> If not defined (default), it equals the <i>contact_url</i>
         * attribute.
         */
        public String from_url = null;
        /**
         * Contact URL. If not defined (default), it is formed by
         * sip:local_user@host_address:host_port
         */
        public String contact_url = null;
        /** User's name (used to build the contact_url if not explitely defined) */
        public String username = null;
        /** User's realm. */
        public String realm = null,realm_orig = null;
        /** User's passwd. */
        public String passwd = null;

        // IMS MMTel settings (added by mandrajg)
        /** q value used at registration */
        public String qvalue = null;
        /** MMTel flavor used */
```

```
public boolean mmtel = false;

public boolean pub = false;

/**
 * Path for the 'ua.jar' lib, used to retrive various UA media (gif, wav,
 * etc.) By default, it is used the "lib/ua.jar" folder
 */
public static String ua_jar = "lib/ua.jar";
/**
 * Path for the 'contacts.lst' file where save and load the VisualUA
 * contacts By default, it is used the "config/contacts.lst" folder
 */
public static String contacts_file = "contacts.lst";

/** Whether registering with the registrar server */
public boolean do_register = false;
/** Whether unregistering the contact address */
public boolean do_unregister = false;
/**
 * Whether unregistering all contacts beafore registering the contact
 * address
 */
public boolean do_unregister_all = false;
/** Expires time (in seconds). */
public int expires = 3600;

/**
 * Rate of keep-alive packets sent toward the registrar server (in
 * milliseconds). Set keepalive_time=0 to disable the sending of keep-alive
 * datagrams.
 */
public long keepalive_time = 0;

/**
 * Automatic call a remote user secified by the 'call_to' value. Use value
 * 'NONE' for manual calls (or let it undefined).
 */
public String call_to = null;
/**
 * Automatic answer time in seconds; time<0 corresponds to manual answer
 * mode.
 */
public int accept_time = -1;
/**
 * Automatic hangup time (call duartion) in seconds; time<=0 corresponds to
 * manual hangup mode.
 */
public int hangup_time = -1;
/**
 * Automatic call transfer time in seconds; time<0 corresponds to no auto
 * transfer mode.
 */
public int transfer_time = -1;
/**
 * Automatic re-inviting time in seconds; time<0 corresponds to no auto
 * re-invite mode.
 */
public int re_invite_time = -1;

/**
 * Redirect incoming call to the secified url. Use value 'NONE' for not
 * redirecting incoming calls (or let it undefined).
 */
public String redirect_to = null;

/**
 * Transfer calls to the secified url. Use value 'NONE' for not transferring
 * calls (or let it undefined).
 */
public String transfer_to = null;

/** No offer in the invite */
public boolean no_offer = false;
/** Do not use prompt */
public boolean no_prompt = false;

/** Whether using audio */
public boolean audio = true; // modified
/** Whether using video */
public boolean video = true; // modified

/** Whether playing in receive only mode */
public boolean recv_only = false;
/** Whether playing in send only mode */
public boolean send_only = false;
/** Whether playing a test tone in send only mode */
public boolean send_tone = false;
/** Audio file to be played */
public String send_file = null;
/** Audio file to be recorded */
public String recv_file = null;

/** Audio port */
public int audio_port = 21000;
```

```
public int[] audio_codecs = {3, 8, 0};
public int dtmf_avp = 101; // zero means no use of outband DTMF
/** Audio sample rate */
public int audio_sample_rate = 8000;
/** Audio sample size */
public int audio_sample_size = 1;
/** Audio frame size */
public int audio_frame_size = 160;

/** Video port */
public int video_port = 21070;
/** Video avp */
public int video_avp = 103;

// ************************* Costructors *************************

/** Costructs a void UserProfile */
public UserAgentProfile() {
          init();
}

/** Costructs a new UserAgentProfile */
public UserAgentProfile(String file) { // load configuration
          loadFile(file);
          // post-load manipulation
          init();
}

/** Inits the UserAgentProfile */
private void init() {
          if (realm == null && contact_url != null)
                    realm = new NameAddress(contact_url).getAddress().getHost();
          if (username == null)
                    username = (contact_url != null) ? new NameAddress(contact_url)
                                        .getAddress().getUserName() : "user";
          if (call_to != null && call_to.equalsIgnoreCase(Configure.NONE))
                    call_to = null;
          if (redirect_to != null && redirect_to.equalsIgnoreCase(Configure.NONE))
                    redirect_to = null;
          if (transfer_to != null && transfer_to.equalsIgnoreCase(Configure.NONE))
                    transfer_to = null;
          if (send_file != null && send_file.equalsIgnoreCase(Configure.NONE))
                    send_file = null;
          if (recv_file != null && recv_file.equalsIgnoreCase(Configure.NONE))
                    recv_file = null;
}

// ************************ Public methods ************************

/**
 * Sets contact_url and from_url with transport information. <p/> This
 * method actually sets contact_url and from_url only if they haven't still
 * been explicitly initialized.
 */
public void initContactAddress(SipProvider sip_provider) { // contact_url
          if (contact_url == null) {
                    contact_url = "sip:" + username + "@"
                                        + sip_provider.getViaAddress();
                    if (sip_provider.getPort() != SipStack.default_port)
                              contact_url += ":" + sip_provider.getPort();
                    if (!sip_provider.getDefaultTransport().equals(
                                        SipProvider.PROTO_UDP))
                              contact_url += ";transport="
                                                  + sip_provider.getDefaultTransport();
          }
          // from_url
          if (from_url == null)
                    from_url = contact_url;
}

// ********************** Protected methods **********************

/** Parses a single line (loaded from the config file) */
protected void parseLine(String line) {
          String attribute;
          Parser par;
          int index = line.indexOf("=");
          if (index > 0) {
                    attribute = line.substring(0, index).trim();
                    par = new Parser(line, index + 1);
          } else {
                    attribute = line;
                    par = new Parser("");
          }

          if (attribute.equals("from_url")) {
                    from_url = par.getRemainingString().trim();
                    return;
          }
          if (attribute.equals("contact_url")) {
                    contact_url = par.getRemainingString().trim();
                    return;
          }
          if (attribute.equals("username")) {
                    username = par.getString();
```

```
            return;
}
if (attribute.equals("realm")) {
            realm = par.getRemainingString().trim();
            return;
}
if (attribute.equals("passwd")) {
            passwd = par.getRemainingString().trim();
            return;
}
if (attribute.equals("ua_jar")) {
            ua_jar = par.getStringUnquoted();
            return;
}
if (attribute.equals("contacts_file")) {
            contacts_file = par.getStringUnquoted();
            return;
}

if (attribute.equals("do_register")) {
            do_register = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("do_unregister")) {
            do_unregister = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("do_unregister_all")) {
            do_unregister_all = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("expires")) {
            expires = par.getInt();
            return;
}
if (attribute.equals("keepalive_time")) {
            keepalive_time = par.getInt();
            return;
}

if (attribute.equals("call_to")) {
            call_to = par.getRemainingString().trim();
            return;
}
if (attribute.equals("accept_time")) {
            accept_time = par.getInt();
            return;
}
if (attribute.equals("hangup_time")) {
            hangup_time = par.getInt();
            return;
}
if (attribute.equals("transfer_time")) {
            transfer_time = par.getInt();
            return;
}
if (attribute.equals("re_invite_time")) {
            re_invite_time = par.getInt();
            return;
}
if (attribute.equals("redirect_to")) {
            redirect_to = par.getRemainingString().trim();
            return;
}
if (attribute.equals("transfer_to")) {
            transfer_to = par.getRemainingString().trim();
            return;
}
if (attribute.equals("no_offer")) {
            no_offer = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("no_prompt")) {
            no_prompt = (par.getString().toLowerCase().startsWith("y"));
            return;
}

if (attribute.equals("audio")) {
            audio = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("video")) {
            video = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("recv_only")) {
            recv_only = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("send_only")) {
            send_only = (par.getString().toLowerCase().startsWith("y"));
            return;
}
if (attribute.equals("send_tone")) {
            send_tone = (par.getString().toLowerCase().startsWith("y"));
```

```
                        return;
                }
                if (attribute.equals("send_file")) {
                        send_file = par.getRemainingString().trim();
                        return;
                }
                if (attribute.equals("recv_file")) {
                        recv_file = par.getRemainingString().trim();
                        return;
                }

                if (attribute.equals("audio_port")) {
                        audio_port = par.getInt();
                        return;
                }
                if (attribute.equals("audio_sample_rate")) {
                        audio_sample_rate = par.getInt();
                        return;
                }
                if (attribute.equals("audio_sample_size")) {
                        audio_sample_size = par.getInt();
                        return;
                }
                if (attribute.equals("audio_frame_size")) {
                        audio_frame_size = par.getInt();
                        return;
                }
                if (attribute.equals("video_port")) {
                        video_port = par.getInt();
                        return;
                }
                if (attribute.equals("video_avp")) {
                        video_avp = par.getInt();
                        return;
                }

                // for backward compatibily
                if (attribute.equals("contact_user")) {
                        username = par.getString();
                        return;
                }
                if (attribute.equals("auto_accept")) {
                        accept_time = ((par.getString().toLowerCase().startsWith("y"))) ? 0
                                        : -1;
                        return;
                }
        }

        /** Converts the entire object into lines (to be saved into the config file) */
        protected String toLines() { // currently not implemented..
                return contact_url;
        }

}
```

➢ /Sipdroid/src/org/sipdroid/sipua/phone

ButtonGridLayout.java

```
package org.sipdroid.sipua.phone;
import android.content.Context;
import android.util.AttributeSet;
import android.view.View;
import android.view.ViewGroup;

public class ButtonGridLayout extends ViewGroup {

    private final int mColumns = 3;
    private int mPaddingBottom = 0,mPaddingLeft = 0,mPaddingRight = 0,mPaddingTop = 0;

    public ButtonGridLayout(Context context) {
        super(context);
    }

    public ButtonGridLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public ButtonGridLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        int y = mPaddingTop;
        final int rows = getRows();
        final View child0 = getChildAt(0);
        final int yInc = (getHeight() - mPaddingTop - mPaddingBottom) / rows;
        final int xInc = (getWidth() - mPaddingLeft - mPaddingRight) / mColumns;
        final int childWidth = child0.getMeasuredWidth();
        final int childHeight = child0.getMeasuredHeight();
        final int xOffset = (xInc - childWidth) / 2;
```

```
            final int yOffset = (yInc - childHeight) / 2;

            for (int row = 0; row < rows; row++) {
                int x = mPaddingLeft;
                for (int col = 0; col < mColumns; col++) {
                    int cell = row * mColumns + col;
                    if (cell >= getChildCount()) {
                        break;
                    }
                    View child = getChildAt(cell);
                    child.layout(x + xOffset, y + yOffset,
                            x + xOffset + childWidth,
                            y + yOffset + childHeight);
                    x += xInc;
                }
                y += yInc;
            }
        }

        private int getRows() {
            return (getChildCount() + mColumns - 1) / mColumns;
        }

        @Override
        protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
            int width = mPaddingLeft + mPaddingRight;
            int height = mPaddingTop + mPaddingBottom;

            // Measure the first child and get it's size
            View child = getChildAt(0);
            child.measure(MeasureSpec.UNSPECIFIED , MeasureSpec.UNSPECIFIED);
            int childWidth = child.getMeasuredWidth();
            int childHeight = child.getMeasuredHeight();
            // Make sure the other children are measured as well, to initialize
            for (int i = 1; i < getChildCount(); i++) {
                getChildAt(0).measure(MeasureSpec.UNSPECIFIED , MeasureSpec.UNSPECIFIED);
            }
            // All cells are going to be the size of the first child
            width += mColumns * childWidth;
            height += getRows() * childHeight;

            width = resolveSize(width, widthMeasureSpec);
            height = resolveSize(height, heightMeasureSpec);
            setMeasuredDimension(width, height);
        }

    }
```

Call.java

```
package org.sipdroid.sipua.phone;


public class Call {
    /* Enums */

    public enum State {
        IDLE, ACTIVE, HOLDING, DIALING, ALERTING, INCOMING, WAITING, DISCONNECTED;

        public boolean isAlive() {
            return !(this == IDLE || this == DISCONNECTED);
        }

        public boolean isRinging() {
            return this == INCOMING || this == WAITING;
        }

        public boolean isDialing() {
            return this == DIALING || this == ALERTING;
        }
    }
    State mState = State.IDLE;
    Connection earliest;
    public long base;

    /* Instance Methods */

    /** Do not modify the List result!!! This list is not yours to keep
     *  It will change across event loop iterations            top
     */

    public State getState() {
            return mState;
    }
    public void setState(State state) {
            mState = state;
    }
    public void setConn(Connection conn) {
            earliest = conn;
    }

    /**
     * hasConnections
     * @return true if the call contains one or more connections
     */
```

```
 public boolean hasConnections() {
        return true;
 }

 /**
  * isIdle
  *
  * FIXME rename
  * @return true if the call contains only disconnected connections (if any)
  */
 public boolean isIdle() {
     return !getState().isAlive();
 }

 /**
  * Returns the Connection associated with this Call that was created
  * first, or null if there are no Connections in this Call
  */
 public Connection
 getEarliestConnection() {
     return earliest;
 }

 public boolean
 isDialingOrAlerting() {
     return getState().isDialing();
 }

 public boolean
 isRinging() {
     return getState().isRinging();
 }

}
```

Callcard.java

```
package org.sipdroid.sipua.phone;

/*
 * Copyright (C) 2009 The Sipdroid Open Source Project
 * Copyright (C) 2006 The Android Open Source Project
 *
 * This file is part of Sipdroid (http://www.sipdroid.org)
 *
 * Sipdroid is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This source code is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this source code; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

//import com.android.internal.telephony.Call;
//import com.android.internal.telephony.CallerInfo;
//import com.android.internal.telephony.CallerInfoAsyncQuery;
//import com.android.internal.telephony.Connection;
//import com.android.internal.telephony.Phone;

import android.content.ContentUris;
import android.content.Context;
import android.content.res.Configuration;
import android.graphics.drawable.Drawable;
import android.net.Uri;
//import android.pim.ContactsAsyncHelper;
//import android.pim.DateUtils;
import android.provider.Contacts.People;
import android.text.TextUtils;
import android.util.AttributeSet;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Chronometer;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.TextView;
import org.sipdroid.sipua.*;
import org.sipdroid.sipua.ui.Receiver;

/**
 * "Call card" UI element: the in-call screen contains a tiled layout of call
 * cards, each representing the state of a current "call" (ie. an active call,
 * a call on hold, or an incoming call.)
 */
public class CallCard extends FrameLayout
        implements CallerInfoAsyncQuery.OnQueryCompleteListener,
                ContactsAsyncHelper.OnImageLoadCompleteListener{
```

```
private static final String LOG_TAG = "PHONE/CallCard";
private static final boolean DBG = false;

// Top-level subviews of the CallCard
private ViewGroup mMainCallCard;
private ViewGroup mOtherCallOngoingInfoArea;
private ViewGroup mOtherCallOnHoldInfoArea;

// "Upper" and "lower" title widgets
private TextView mUpperTitle;
private ViewGroup mLowerTitleViewGroup;
private TextView mLowerTitle;
private ImageView mLowerTitleIcon;
public Chronometer mElapsedTime;

// Text colors, used with the lower title and "other call" info areas
private int mTextColorConnected;
private int mTextColorEnded;
private int mTextColorOnHold;

private ImageView mPhoto;
private TextView mName;
private TextView mPhoneNumber;
private TextView mLabel;

// "Other call" info area
private TextView mOtherCallOngoingName;
private TextView mOtherCallOngoingStatus;
private TextView mOtherCallOnHoldName;
private TextView mOtherCallOnHoldStatus;

// Menu button hint
private TextView mMenuButtonHint;

// Track the state for the photo.
private ContactsAsyncHelper.ImageTracker mPhotoTracker;

// A few hardwired constants used in our screen layout.
// TODO: These should all really come from resources, but that's
// nontrivial; see the javadoc for the ConfigurationHelper class.
// For now, let's at least keep them all here in one place
// rather than sprinkled througout this file.
//
static final int MAIN_CALLCARD_MIN_HEIGHT_LANDSCAPE = 200;
static final int CALLCARD_SIDE_MARGIN_LANDSCAPE = 50;
static final float TITLE_TEXT_SIZE_LANDSCAPE = 22F;  // scaled pixels

public void update(int x,int y,int w,int h) {
    setPadding(0, y, 0, 0);
}

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

    if (DBG) log("CallCard constructor...");
    if (DBG) log("- this = " + this);
    if (DBG) log("- context " + context + ", attrs " + attrs);

    // Inflate the contents of this CallCard, and add it (to ourself) as a child.
    LayoutInflater inflater = LayoutInflater.from(context);
    inflater.inflate(
            R.layout.call_card,  // resource
            this,                // root
            true);

    // create a new object to track the state for the photo.
    mPhotoTracker = new ContactsAsyncHelper.ImageTracker();
}

public void reset() {
    if (DBG) log("reset()...");

    // default to show ACTIVE call style, with empty title and status text
    showCallConnected();
    mUpperTitle.setText("");
}

@Override
protected void onFinishInflate() {
    super.onFinishInflate();

    if (DBG) log("CallCard onFinishInflate(this = " + this + ")...");

    mMainCallCard = (ViewGroup) findViewById(R.id.mainCallCard);
    mOtherCallOngoingInfoArea = (ViewGroup) findViewById(R.id.otherCallOngoingInfoArea);
    mOtherCallOnHoldInfoArea = (ViewGroup) findViewById(R.id.otherCallOnHoldInfoArea);

    // "Upper" and "lower" title widgets
    mUpperTitle = (TextView) findViewById(R.id.upperTitle);
    mLowerTitleViewGroup = (ViewGroup) findViewById(R.id.lowerTitleViewGroup);
    mLowerTitle = (TextView) findViewById(R.id.lowerTitle);
    mLowerTitleIcon = (ImageView) findViewById(R.id.lowerTitleIcon);
    mElapsedTime = (Chronometer) findViewById(R.id.elapsedTime);

    // Text colors
```

```
    mTextColorConnected = getResources().getColor(R.color.incall_textConnected);
    mTextColorEnded = getResources().getColor(R.color.incall_textEnded);
    mTextColorOnHold = getResources().getColor(R.color.incall_textOnHold);

    // "Caller info" area, including photo / name / phone numbers / etc
    mPhoto = (ImageView) findViewById(R.id.photo);
    mName = (TextView) findViewById(R.id.name);
    mPhoneNumber = (TextView) findViewById(R.id.phoneNumber);
    mLabel = (TextView) findViewById(R.id.label);

    // "Other call" info area
    mOtherCallOngoingName = (TextView) findViewById(R.id.otherCallOngoingName);
    mOtherCallOngoingStatus = (TextView) findViewById(R.id.otherCallOngoingStatus);
    mOtherCallOnHoldName = (TextView) findViewById(R.id.otherCallOnHoldName);
    mOtherCallOnHoldStatus = (TextView) findViewById(R.id.otherCallOnHoldStatus);

    // Menu Button hint
    mMenuButtonHint = (TextView) findViewById(R.id.menuButtonHint);
}

void updateState(Phone phone) {
    if (DBG) log("updateState(" + phone + ")...");

    // Update some internal state based on the current state of the phone.
    // TODO: This code, and updateForegroundCall() / updateRingingCall(),
    // can probably still be simplified some more.

    Phone.State state = phone.getState();  // IDLE, RINGING, or OFFHOOK
    if (state == Phone.State.RINGING) {
        // A phone call is ringing *or* call waiting
        // (ie. another call may also be active as well.)
        updateRingingCall(phone);
    } else if (state == Phone.State.OFFHOOK) {
        // The phone is off hook. At least one call exists that is
        // dialing, active, or holding, and no calls are ringing or waiting.
        updateForegroundCall(phone);
    } else {
        // Presumably IDLE:  no phone activity
        // TODO: Should we ever be in this state in the first place?
        // (Is there ever any reason to draw the in-call screen
        // if the phone is totally idle?)
        // ==> Possibly during the "call ended" state, for 5 seconds
        //     *after* a call ends...
        // For now:
        Log.w(LOG_TAG, "CallCard updateState: overall Phone state is " + state);
        updateForegroundCall(phone);
    }
}

private void updateForegroundCall(Phone phone) {
    if (DBG) log("updateForegroundCall()...");

    Call fgCall = phone.getForegroundCall();
    Call bgCall = phone.getBackgroundCall();

    if (fgCall.isIdle() && !fgCall.hasConnections()) {
        if (DBG) log("updateForegroundCall: no active call, show holding call");
        // TODO: make sure this case agrees with the latest UI spec.

        // Display the background call in the main info area of the
        // CallCard, since there is no foreground call.  Note that
        // displayMainCallStatus() will notice if the call we passed in is on
        // hold, and display the "on hold" indication.
        fgCall = bgCall;

        // And be sure to not display anything in the "on hold" box.
        bgCall = null;
    }

    displayMainCallStatus(phone, fgCall);
    displayOnHoldCallStatus(phone, bgCall);
    displayOngoingCallStatus(phone, null);
}

private void updateRingingCall(Phone phone) {
    if (DBG) log("updateRingingCall()...");

    Call ringingCall = phone.getRingingCall();
    Call fgCall = phone.getForegroundCall();
    Call bgCall = phone.getBackgroundCall();

    displayMainCallStatus(phone, ringingCall);
    displayOnHoldCallStatus(phone, bgCall);
    displayOngoingCallStatus(phone, fgCall);
}

/**
 * Updates the main block of caller info on the CallCard
 * (ie. the stuff in the mainCallCard block) based on the specified Call.
 */
public void displayMainCallStatus(Phone phone, Call call) {
    if (DBG) log("displayMainCallStatus(phone " + phone
                 + ", call " + call + ", state" + call.getState() + ")...");

    Call.State state = call.getState();
```

```
        int callCardBackgroundResid = 0;

        // Background frame resources are different between portrait/landscape:
        boolean landscapeMode = getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE;

        switch (state) {
            case ACTIVE:
                showCallConnected();

                callCardBackgroundResid =
                        landscapeMode ? R.drawable.incall_frame_connected_tall_land
                        : R.drawable.incall_frame_connected_tall_port;

                // update timer field
                if (DBG) log("displayMainCallStatus: start periodicUpdateTimer");
                break;

            case HOLDING:
                showCallOnhold();

                callCardBackgroundResid =
                        landscapeMode ? R.drawable.incall_frame_hold_tall_land
                        : R.drawable.incall_frame_hold_tall_port;
                break;

            case DISCONNECTED:
                reset();
                showCallEnded();

                callCardBackgroundResid =
                        landscapeMode ? R.drawable.incall_frame_ended_tall_land
                        : R.drawable.incall_frame_ended_tall_port;

                break;

            case DIALING:
            case ALERTING:
                showCallConnecting();

                callCardBackgroundResid =
                        landscapeMode ? R.drawable.incall_frame_normal_tall_land
                        : R.drawable.incall_frame_normal_tall_port;

                break;

            case INCOMING:
            case WAITING:
                showCallIncoming();

                callCardBackgroundResid =
                        landscapeMode ? R.drawable.incall_frame_normal_tall_land
                        : R.drawable.incall_frame_normal_tall_port;
               break;

            case IDLE:
                // The "main CallCard" should never display an idle call!
                Log.w(LOG_TAG, "displayMainCallStatus: IDLE call in the main call card!");
                break;

            default:
                Log.w(LOG_TAG, "displayMainCallStatus: unexpected call state: " + state);
                break;
        }

        updateCardTitleWidgets(phone, call);

    {
            Connection conn = call.getEarliestConnection();

            boolean isPrivateNumber = false; // TODO: need isPrivate() API

            if (conn == null) {
                if (DBG) log("displayMainCallStatus: connection is null, using default values.");
                // if the connection is null, we run through the behaviour
                // we had in the past, which breaks down into trivial steps
                // with the current implementation of getCallerInfo and
                // updateDisplayForPerson.
                updateDisplayForPerson(null, isPrivateNumber, false, call);
            } else {
                if (DBG) log("  - CONN: " + conn + ", state = " + conn.getState());

                // make sure that we only make a new query when the current
                // callerinfo differs from what we've been requested to display.
                boolean runQuery = true;
                Object o = conn.getUserData();
                if (o instanceof PhoneUtils.CallerInfoToken) {
                    runQuery = mPhotoTracker.isDifferentImageRequest(
                            ((PhoneUtils.CallerInfoToken) o).currentInfo);
                } else {
                    runQuery = mPhotoTracker.isDifferentImageRequest(conn);
                }

                if (runQuery) {
                    if (DBG) log("- displayMainCallStatus: starting CallerInfo query...");
                    PhoneUtils.CallerInfoToken info =
```

```
                    PhoneUtils.startGetCallerInfo(getContext(), conn, this, call);
                updateDisplayForPerson(info.currentInfo, isPrivateNumber, !info.isFinal, call);
            } else {
                // No need to fire off a new query.  We do still need
                // to update the display, though (since we might have
                // previously been in the "conference call" state.)
                if (DBG) log("- displayMainCallStatus: using data we already have...");
                if (o instanceof CallerInfo) {
                    CallerInfo ci = (CallerInfo) o;
                    if (DBG) log("   ==> Got CallerInfo; updating display: ci = " + ci);
                    updateDisplayForPerson(ci, false, false, call);
                } else if (o instanceof PhoneUtils.CallerInfoToken){
                    CallerInfo ci = ((PhoneUtils.CallerInfoToken) o).currentInfo;
                    if (DBG) log("   ==> Got CallerInfoToken; updating display: ci = " + ci);
                    updateDisplayForPerson(ci, false, true, call);
                } else {
                    Log.w(LOG_TAG, "displayMainCallStatus: runQuery was false, "
                          + "but we didn't have a cached CallerInfo object!  o = " + o);
                    // TODO: any easy way to recover here (given that
                    // the CallCard is probably displaying stale info
                    // right now?)  Maybe force the CallCard into the
                    // "Unknown" state?
                }
            }
        }
    }


    updatePhotoForCallState(call);

    // Set the background frame color based on the state of the call.
    setMainCallCardBackgroundResource(callCardBackgroundResid);
    // (Text colors are set in updateCardTitleWidgets().)
}

/**
 * Implemented for CallerInfoAsyncQuery.OnQueryCompleteListener interface.
 * refreshes the CallCard data when it called.
 */
public void onQueryComplete(int token, Object cookie, CallerInfo ci) {
    if (DBG) log("onQueryComplete: token " + token + ", cookie " + cookie + ", ci " + ci);

    if (cookie instanceof Call) {
        if (DBG) log("callerinfo query complete, updating ui from displayMainCallStatus()");
        Call call = (Call) cookie;
        updateDisplayForPerson(ci, false, false, call);
        updatePhotoForCallState(call);

    } else if (cookie instanceof TextView){
        if (DBG) log("callerinfo query complete, updating ui from ongoing or onhold");
        ((TextView) cookie).setText(PhoneUtils.getCompactNameFromCallerInfo(ci, getContext()));
    }
}

/**
 * Implemented for ContactsAsyncHelper.OnImageLoadCompleteListener interface.
 * make sure that the call state is reflected after the image is loaded.
 */
public void onImageLoadComplete(int token, Object cookie, ImageView iView,
        boolean imagePresent){
    if (cookie != null) {
        updatePhotoForCallState((Call) cookie);
    }
}

/**
 * Updates the "upper" and "lower" titles based on the current state of this call.
 */
private void updateCardTitleWidgets(Phone phone, Call call) {
    if (DBG) log("updateCardTitleWidgets(call " + call + ")...");
    Call.State state = call.getState();
    String cardTitle = getTitleForCallCard(call);

    if (DBG) log("updateCardTitleWidgets: " + cardTitle);

    // We display *either* the "upper title" or the "lower title", but
    // never both.
    if (state == Call.State.ACTIVE) {
        // Use the "lower title" (in green).
        mLowerTitleViewGroup.setVisibility(View.VISIBLE);
        mLowerTitleIcon.setImageResource(R.drawable.ic_incall_ongoing);
        mLowerTitle.setText(cardTitle);
        mLowerTitle.setTextColor(mTextColorConnected);
        mElapsedTime.setTextColor(mTextColorConnected);
        mElapsedTime.setBase(call.base);
        mElapsedTime.start();
        mElapsedTime.setVisibility(View.VISIBLE);
        mUpperTitle.setText("");
    } else if (state == Call.State.DISCONNECTED) {
        // Use the "lower title" (in red).
        // TODO: We may not *always* want to use the lower title for
        // the DISCONNECTED state.  "Error" states like BUSY or
        // CONGESTION (see getCallFailedString()) should probably go
        // in the upper title, for example.  In fact, the lower title
        // should probably be used *only* for the normal "Call ended"
```

```
        // case.
        mLowerTitleViewGroup.setVisibility(View.VISIBLE);
        mLowerTitleIcon.setImageResource(R.drawable.ic_incall_end);
        mLowerTitle.setText(cardTitle);
        mLowerTitle.setTextColor(mTextColorEnded);
        mElapsedTime.setTextColor(mTextColorEnded);
        if (call.base != 0) {
                mElapsedTime.setBase(call.base);
                mElapsedTime.start();
                mElapsedTime.stop();
        } else
                mElapsedTime.setVisibility(View.INVISIBLE);
        mUpperTitle.setText("");
    } else {
        // All other states use the "upper title":
        mUpperTitle.setText(cardTitle);
        mLowerTitleViewGroup.setVisibility(View.INVISIBLE);
        if (state != Call.State.HOLDING)
               mElapsedTime.setVisibility(View.INVISIBLE);
    }
}

 /**
 * Returns the "card title" displayed at the top of a foreground
 * ("active") CallCard to indicate the current state of this call, like
 * "Dialing" or "In call" or "On hold".  A null return value means that
 * there's no title string for this state.
 */
private String getTitleForCallCard(Call call) {
    String retVal = null;
    Call.State state = call.getState();
    Context context = getContext();

    if (DBG) log("- getTitleForCallCard(Call " + call + ")...");

    switch (state) {
        case IDLE:
            break;

        case ACTIVE:
            // Title is "Call in progress".  (Note this appears in the
            // "lower title" area of the CallCard.)
            retVal = context.getString(R.string.card_title_in_progress);
            break;

        case HOLDING:
            retVal = context.getString(R.string.card_title_on_hold);
            // TODO: if this is a conference call on hold,
            // maybe have a special title here too?
            break;

        case DIALING:
        case ALERTING:
            retVal = context.getString(R.string.card_title_dialing);
            break;

        case INCOMING:
        case WAITING:
            retVal = context.getString(R.string.card_title_incoming_call);
            break;

        case DISCONNECTED:
            retVal = getCallFailedString(call);
            break;
    }

    if (DBG) log("  ==> result: " + retVal);
    return retVal;
}

/**
 * Updates the "on hold" box in the "other call" info area
 * (ie. the stuff in the otherCallOnHoldInfo block)
 * based on the specified Call.
 * Or, clear out the "on hold" box if the specified call
 * is null or idle.
 */
public void displayOnHoldCallStatus(Phone phone, Call call) {
    if (DBG) log("displayOnHoldCallStatus(call =" + call + ")...");
    if (call == null) {
        mOtherCallOnHoldInfoArea.setVisibility(View.GONE);
        return;
    }

    Call.State state = call.getState();
    switch (state) {
        case HOLDING:

                if (DBG) log("==> NOT a conf call; call startGetCallerInfo...");
                PhoneUtils.CallerInfoToken info = PhoneUtils.startGetCallerInfo(
                        getContext(), call, this, mOtherCallOnHoldName);
                name = PhoneUtils.getCompactNameFromCallerInfo(info.currentInfo, getContext());
            }

            mOtherCallOnHoldName.setText(name);
```

```
                // The call here is always "on hold", so use the orange "hold" frame
                // and orange text color:
                setOnHoldInfoAreaBackgroundResource(R.drawable.incall_frame_hold_short);
                mOtherCallOnHoldName.setTextColor(mTextColorOnHold);
                mOtherCallOnHoldStatus.setTextColor(mTextColorOnHold);

                mOtherCallOnHoldInfoArea.setVisibility(View.VISIBLE);

                break;

            default:
                // There's actually no call on hold.  (Presumably this call's
                // state is IDLE, since any other state is meaningless for the
                // background call.)
                mOtherCallOnHoldInfoArea.setVisibility(View.GONE);
                break;
        }
    }

    /**
     * Updates the "Ongoing call" box in the "other call" info area
     * (ie. the stuff in the otherCallOngoingInfo block)
     * based on the specified Call.
     * Or, clear out the "ongoing call" box if the specified call
     * is null or idle.
     */
    public void displayOngoingCallStatus(Phone phone, Call call) {
        if (DBG) log("displayOngoingCallStatus(call =" + call + ")...");
        if (call == null) {
            mOtherCallOngoingInfoArea.setVisibility(View.GONE);
            return;
        }

        Call.State state = call.getState();
        switch (state) {
            case ACTIVE:
            case DIALING:
            case ALERTING:
                // Ok, there actually is an ongoing call.
                // Display the "ongoing call" box.
                String name;

                // First, see if we need to query.
{
                    // perform query and update the name temporarily
                    // make sure we hand the textview we want updated to the
                    // callback function.
                    PhoneUtils.CallerInfoToken info = PhoneUtils.startGetCallerInfo(
                            getContext(), call, this, mOtherCallOngoingName);
                    name = PhoneUtils.getCompactNameFromCallerInfo(info.currentInfo, getContext());
                }

                mOtherCallOngoingName.setText(name);

                // The call here is always "ongoing", so use the green "connected" frame
                // and green text color:
                setOngoingInfoAreaBackgroundResource(R.drawable.incall_frame_connected_short);
                mOtherCallOngoingName.setTextColor(mTextColorConnected);
                mOtherCallOngoingStatus.setTextColor(mTextColorConnected);

                mOtherCallOngoingInfoArea.setVisibility(View.VISIBLE);

                break;

            default:
                // There's actually no ongoing call.  (Presumably this call's
                // state is IDLE, since any other state is meaningless for the
                // foreground call.)
                mOtherCallOngoingInfoArea.setVisibility(View.GONE);
                break;
        }
    }


    private String getCallFailedString(Call call) {
        int resID = R.string.card_title_call_ended;

        if (Receiver.call_end_reason != -1)
            resID = Receiver.call_end_reason;

        return getContext().getString(resID);
    }

    private void showCallConnecting() {
        if (DBG) log("showCallConnecting()...");
        // TODO: remove if truly unused
    }

    private void showCallIncoming() {
        if (DBG) log("showCallIncoming()...");
        // TODO: remove if truly unused
    }

    private void showCallConnected() {
```

```
        if (DBG) log("showCallConnected()...");
        // TODO: remove if truly unused
    }

    private void showCallEnded() {
        if (DBG) log("showCallEnded()...");
        // TODO: remove if truly unused
    }
    private void showCallOnhold() {
        if (DBG) log("showCallOnhold()...");
        // TODO: remove if truly unused
    }

    /**
     * Updates the name / photo / number / label fields on the CallCard
     * based on the specified CallerInfo.
     *
     * If the current call is a conference call, use
     * updateDisplayForConference() instead.
     */
    private void updateDisplayForPerson(CallerInfo info,
                                        boolean isPrivateNumber,
                                        boolean isTemporary,
                                        Call call) {
        if (DBG) log("updateDisplayForPerson(" + info + ")...");

        // inform the state machine that we are displaying a photo.
        mPhotoTracker.setPhotoRequest(info);
        mPhotoTracker.setPhotoState(ContactsAsyncHelper.ImageTracker.DISPLAY_IMAGE);

        String name;
        String displayNumber = null;
        String label = null;
        Uri personUri = null;

        if (info != null) {
            if (TextUtils.isEmpty(info.name)) {
                if (TextUtils.isEmpty(info.phoneNumber)) {
                   {
                        name = getContext().getString(R.string.unknown);
                    }
                } else {
                     name = info.phoneNumber;
                }
            } else {
                name = info.name;
                displayNumber = info.phoneNumber;
                label = info.phoneLabel;
            }
            personUri = ContentUris.withAppendedId(People.CONTENT_URI, info.person_id);
        } else {
            {
                name = getContext().getString(R.string.unknown);
            }
        }
        mName.setText(name);
        mName.setVisibility(View.VISIBLE);
     if (isTemporary && (info == null || !info.isCachedPhotoCurrent)) {
            mPhoto.setVisibility(View.INVISIBLE);
        } else if (info != null && info.photoResource != 0){
            showImage(mPhoto, info.photoResource);
        } else if (!showCachedImage(mPhoto, info)) {
            // Load the image with a callback to update the image state.
            // Use a placeholder image value of -1 to indicate no image.
            ContactsAsyncHelper.updateImageViewWithContactPhotoAsync(info, 0, this, call,
                    getContext(), mPhoto, personUri, -1);
        }
        if (displayNumber != null) {
            mPhoneNumber.setText(displayNumber);
            mPhoneNumber.setVisibility(View.VISIBLE);
        } else {
//            mPhoneNumber.setVisibility(View.GONE);
        mPhoneNumber.setText("");
        }

        if (label != null) {
            mLabel.setText(label);
            mLabel.setVisibility(View.VISIBLE);
        } else {
//            mLabel.setVisibility(View.GONE);
        mLabel.setText("");
        }
    }

    /**
     * Updates the CallCard "photo" IFF the specified Call is in a state
     * that needs a special photo (like "busy" or "dialing".)
     *
     * If the current call does not require a special image in the "photo"
     * slot onscreen, don't do anything, since presumably the photo image
     * has already been set (to the photo of the person we're talking, or
     * the generic "picture_unknown" image, or the "conference call"
     * image.)
     */
    private void updatePhotoForCallState(Call call) {
```

```
        if (DBG) log("updatePhotoForCallState(" + call + ")...");
        int photoImageResource = 0;

        // Check for the (relatively few) telephony states that need a
        // special image in the "photo" slot.
        Call.State state = call.getState();
        switch (state) {
            case DISCONNECTED:
                // Display the special "busy" photo for BUSY or CONGESTION.
                // Otherwise (presumably the normal "call ended" state)
                // leave the photo alone.
                Connection c = call.getEarliestConnection();
                // if the connection is null, we assume the default case,
                // otherwise update the image resource normally.
                if (c != null) {
                    Connection.DisconnectCause cause = c.getDisconnectCause();
                    if ((cause == Connection.DisconnectCause.BUSY)
                        || (cause == Connection.DisconnectCause.CONGESTION)) {
                        photoImageResource = R.drawable.picture_busy;
                    }
                } else if (DBG) {
                    log("updatePhotoForCallState: connection is null, ignoring.");
                }

                // TODO: add special images for any other DisconnectCauses?
                break;

            case DIALING:
            case ALERTING:
                photoImageResource = R.drawable.picture_dialing;
                break;

            default:
                CallerInfo ci = null;
                {
                    Connection conn = call.getEarliestConnection();
                    if (conn != null) {
                        Object o = conn.getUserData();
                        if (o instanceof CallerInfo) {
                            ci = (CallerInfo) o;
                        } else if (o instanceof PhoneUtils.CallerInfoToken) {
                            ci = ((PhoneUtils.CallerInfoToken) o).currentInfo;
                        }
                    }
                }

                if (ci != null) {
                    photoImageResource = ci.photoResource;
                }

                if (photoImageResource == 0) {
{
                        if (!showCachedImage(mPhoto, ci) && (mPhotoTracker.getPhotoState() ==
                                ContactsAsyncHelper.ImageTracker.DISPLAY_DEFAULT)) {
                            ContactsAsyncHelper.updateImageViewWithContactPhotoAsync(ci,
                                    getContext(), mPhoto, mPhotoTracker.getPhotoUri(), -1);
                            mPhotoTracker.setPhotoState(
                                    ContactsAsyncHelper.ImageTracker.DISPLAY_IMAGE);
                        }
                    }
                } else {
                    showImage(mPhoto, photoImageResource);
                    mPhotoTracker.setPhotoState(ContactsAsyncHelper.ImageTracker.DISPLAY_IMAGE);
                    return;
                }
                break;
        }

        if (photoImageResource != 0) {
            if (DBG) log("- overrriding photo image: " + photoImageResource);
            showImage(mPhoto, photoImageResource);
            // Track the image state.
            mPhotoTracker.setPhotoState(ContactsAsyncHelper.ImageTracker.DISPLAY_DEFAULT);
        }
    }

    /**
     * Try to display the cached image from the callerinfo object.
     *
     *  @return true if we were able to find the image in the cache, false otherwise.
     */
    private static final boolean showCachedImage (ImageView view, CallerInfo ci) {
        if ((ci != null) && ci.isCachedPhotoCurrent) {
            if (ci.cachedPhoto != null) {
                showImage(view, ci.cachedPhoto);
            } else {
                showImage(view, R.drawable.picture_unknown);
            }
            return true;
        }
        return false;
    }

    /** Helper function to display the resource in the imageview AND ensure its visibility.*/
    private static final void showImage(ImageView view, int resource) {
```

```
    view.setImageResource(resource);
    view.setVisibility(View.VISIBLE);
}

/** Helper function to display the drawable in the imageview AND ensure its visibility.*/
private static final void showImage(ImageView view, Drawable drawable) {
    view.setImageDrawable(drawable);
    view.setVisibility(View.VISIBLE);
}

private SlidingCardManager mSlidingCardManager;

@Override
public boolean dispatchTouchEvent(MotionEvent ev) {
    if (mSlidingCardManager != null) mSlidingCardManager.handleCallCardTouchEvent(ev);
    return true;
}

public void setSlidingCardManager(SlidingCardManager slidingCardManager) {
    mSlidingCardManager = slidingCardManager;
}

/**
 * Sets the background drawable of the main call card.
 */
private void setMainCallCardBackgroundResource(int resid) {
    mMainCallCard.setBackgroundResource(resid);
}

/**
 * Sets the background drawable of the "ongoing call" info area.
 */
private void setOngoingInfoAreaBackgroundResource(int resid) {
    mOtherCallOngoingInfoArea.setBackgroundResource(resid);
}

/**
 * Sets the background drawable of the "call on hold" info area.
 */
private void setOnHoldInfoAreaBackgroundResource(int resid) {
    mOtherCallOnHoldInfoArea.setBackgroundResource(resid);
}

/**
 * Returns the "Menu button hint" TextView (which is manipulated
 * directly by the InCallScreen.)
 * @see InCallScreen.updateMenuButtonHint()
 */
public /* package */ TextView getMenuButtonHint() {
    return mMenuButtonHint;
}

/**
 * Updates anything about our View hierarchy or internal state
 * that needs to be different in landscape mode.
 *
 * @see InCallScreen.applyConfigurationToLayout()
 */
/* package */ public void updateForLandscapeMode() {
    if (DBG) log("updateForLandscapeMode()...");

    // The main CallCard's minimum height is smaller in landscape mode
    // than in portrait mode.
    mMainCallCard.setMinimumHeight(MAIN_CALLCARD_MIN_HEIGHT_LANDSCAPE);

    // Add some left and right margin to the top-level elements, since
    // there's no need to use the full width of the screen (which is
    // much wider in landscape mode.)
    setSideMargins(mMainCallCard, CALLCARD_SIDE_MARGIN_LANDSCAPE);
    setSideMargins(mOtherCallOngoingInfoArea, CALLCARD_SIDE_MARGIN_LANDSCAPE);
    setSideMargins(mOtherCallOnHoldInfoArea, CALLCARD_SIDE_MARGIN_LANDSCAPE);

    // A couple of TextViews are slightly smaller in landscape mode.
    mUpperTitle.setTextSize(TITLE_TEXT_SIZE_LANDSCAPE);
}

/**
 * Sets the left and right margins of the specified ViewGroup (whose
 * LayoutParams object which must inherit from
 * ViewGroup.MarginLayoutParams.)
 *
 * TODO: Is there already a convenience method like this somewhere?
 */
private void setSideMargins(ViewGroup vg, int margin) {
    ViewGroup.MarginLayoutParams lp =
            (ViewGroup.MarginLayoutParams) vg.getLayoutParams();
    // Equivalent to setting android:layout_marginLeft/Right in XML
    lp.leftMargin = margin;
    lp.rightMargin = margin;
    vg.setLayoutParams(lp);
}


// Debugging / testing code
```

```
        private void log(String msg) {
            Log.d(LOG_TAG, "[CallCard " + this + "] " + msg);
        }
    }
```

CallerInfo.java

```
package org.sipdroid.sipua.phone;
import android.content.Context;
import android.database.Cursor;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.provider.Contacts;
import android.provider.Contacts.People;
import android.provider.Contacts.Phones;
import android.util.Config;
import android.util.Log;

/**
 * Looks up caller information for the given phone number.
 *
 * {@hide}
 */
public class CallerInfo {
    private static final String TAG = "CallerInfo";

    public static final String UNKNOWN_NUMBER = "-1";
    public static final String PRIVATE_NUMBER = "-2";
    public String name;
    public String phoneNumber;
    public String phoneLabel;
    /* Split up the phoneLabel into number type and label name */
    public int    numberType;
    public String numberLabel;

    public int photoResource;
    public long person_id;
    public boolean needUpdate;
    public Uri contactRefUri;

    // fields to hold individual contact preference data,
    // including the send to voicemail flag and the ringtone
    // uri reference.
    public Uri contactRingtoneUri;
    public boolean shouldSendToVoicemail;

    /**
     * Drawable representing the caller image.  This is essentially
     * a cache for the image data tied into the connection /
     * callerinfo object.  The isCachedPhotoCurrent flag indicates
     * if the image data needs to be reloaded.
     */
    public Drawable cachedPhoto;
    public boolean isCachedPhotoCurrent;

    public CallerInfo() {
    }

    /**
     * getCallerInfo given a Cursor.
     * @param context the context used to retrieve string constants
     * @param contactRef the URI to attach to this CallerInfo object
     * @param cursor the first object in the cursor is used to build the CallerInfo object.
     * @return the CallerInfo which contains the caller id for the given
     * number. The returned CallerInfo is null if no number is supplied.
     */
    public static CallerInfo getCallerInfo(Context context, Uri contactRef, Cursor cursor) {

        CallerInfo info = new CallerInfo();
        info.photoResource = 0;
        info.phoneLabel = null;
        info.numberType = 0;
        info.numberLabel = null;
        info.cachedPhoto = null;
        info.isCachedPhotoCurrent = false;

        if (Config.LOGV) Log.v(TAG, "construct callerInfo from cursor");

        if (cursor != null) {
            if (cursor.moveToFirst()) {

                int columnIndex;

                // Look for the name
                columnIndex = cursor.getColumnIndex(People.NAME);
                if (columnIndex != -1) {
                    info.name = cursor.getString(columnIndex);
                }

                // Look for the number
                columnIndex = cursor.getColumnIndex(Phones.NUMBER);
                if (columnIndex != -1) {
                    info.phoneNumber = cursor.getString(columnIndex);
                }
```

```
                // Look for the label/type combo
                columnIndex = cursor.getColumnIndex(Phones.LABEL);
                if (columnIndex != -1) {
                    int typeColumnIndex = cursor.getColumnIndex(Phones.TYPE);
                    if (typeColumnIndex != -1) {
                        info.numberType = cursor.getInt(typeColumnIndex);
                        info.numberLabel = cursor.getString(columnIndex);
                        info.phoneLabel = Contacts.Phones.getDisplayLabel(context,
                                info.numberType, info.numberLabel)
                                .toString();
                    }
                }

                // Look for the person ID
                columnIndex = cursor.getColumnIndex(Phones.PERSON_ID);
                if (columnIndex != -1) {
                    info.person_id = cursor.getLong(columnIndex);
                } else {
                    columnIndex = cursor.getColumnIndex(People._ID);
                    if (columnIndex != -1) {
                        info.person_id = cursor.getLong(columnIndex);
                    }
                }

                // look for the custom ringtone, create from the string stored
                // in the database.
                columnIndex = cursor.getColumnIndex(People.CUSTOM_RINGTONE);
                if ((columnIndex != -1) && (cursor.getString(columnIndex) != null)) {
                    info.contactRingtoneUri = Uri.parse(cursor.getString(columnIndex));
                } else {
                    info.contactRingtoneUri = null;
                }

                // look for the send to voicemail flag, set it to true only
                // under certain circumstances.
                columnIndex = cursor.getColumnIndex(People.SEND_TO_VOICEMAIL);
                info.shouldSendToVoicemail = (columnIndex != -1) &&
                        ((cursor.getInt(columnIndex)) == 1);
            }
            cursor.close();
        }

        info.needUpdate = false;
        info.name = normalize(info.name);
        info.contactRefUri = contactRef;

        return info;
    }

    /**
     * getCallerInfo given a URI, look up in the call-log database
     * for the uri unique key.
     * @param context the context used to get the ContentResolver
     * @param contactRef the URI used to lookup caller id
     * @return the CallerInfo which contains the caller id for the given
     * number. The returned CallerInfo is null if no number is supplied.
     */
    public static CallerInfo getCallerInfo(Context context, Uri contactRef) {

        return getCallerInfo(context, contactRef,
                context.getContentResolver().query(contactRef, null, null, null, null));
    }

    private static String normalize(String s) {
        if (s == null || s.length() > 0) {
            return s;
        } else {
            return null;
        }
    }
}
```

CallStateException.java

```
package org.sipdroid.sipua.phone;
public class CallStateException extends Exception
{
    public
    CallStateException()
    {
    }

    public
    CallStateException(String string)
    {
        super(string);
    }
}
```

Connection.java

```
package org.sipdroid.sipua.phone;
import org.sipdroid.sipua.ui.Receiver;
import android.content.ContentResolver;
import android.content.ContentValues;
```

```
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.os.SystemClock;
import android.provider.CallLog;
import android.provider.CallLog.Calls;
import android.provider.Contacts.People;
import android.text.TextUtils;
public class Connection
{
    public enum DisconnectCause {
        NOT_DISCONNECTED,   /* has not yet disconnected */
        INCOMING_MISSED,    /* an incoming call that was missed and never answered */
        NORMAL,             /* normal; remote */
        LOCAL,              /* normal; local hangup */
        BUSY,               /* outgoing call to busy line */
        CONGESTION,         /* outgoing call to congested network */
        MMI,                /* not presently used; dial() returns null */
        INVALID_NUMBER,     /* invalid dial string */
        LOST_SIGNAL,
        LIMIT_EXCEEDED,     /* eg GSM ACM limit exceeded */
        INCOMING_REJECTED,  /* an incoming call that was rejected */
        POWER_OFF,          /* radio is turned off explicitly */
        OUT_OF_SERVICE,     /* out of service */
        SIM_ERROR,          /* No SIM, SIM locked, or other SIM error */
        CALL_BARRED,        /* call was blocked by call barrring */
        FDN_BLOCKED         /* call was blocked by fixed dial number */
    }

    /** ACTION for publishing information about calls. */
    private static final String ACTION_CM_SIP = // .
    "de.ub0r.android.callmeter.SAVE_SIPCALL";
    /** Extra holding uri of done call. */
    private static final String EXTRA_SIP_URI = "uri";
    /** Extra holding name of provider. */
    private static final String EXTRA_SIP_PROVIDER = "provider";

    Object userData;

    /* Instance Methods */

    /**
     * Gets address (e.g., phone number) associated with connection
     * TODO: distinguish reasons for unavailablity
     *
     * @return address or null if unavailable
     */
    String address,address2;
    public String getAddress() {
           return address;
    }
    public String getAddress2() {
           return address2;
    }
    public void setAddress(String a,String b) {
           address = a;
           address2 = b;
    }

    Call c;
    /**
     * @return Call that owns this Connection, or null if none
     */
    public Call getCall() {
           return c;
    }
    public void setCall(Call a) {
           c = a;
    }

     /**
     * Returns "NOT_DISCONNECTED" if not yet disconnected
     */
    public DisconnectCause getDisconnectCause() {
           return DisconnectCause.NORMAL;
    }

   /**
     * If this Connection is connected, then it is associated with
     * a Call.
     *
     * Returns getCall().getState() or Call.State.IDLE if not
     * connected
     */
    public Call.State getState()
    {
        Call c;

        c = getCall();

        if (c == null) {
            return Call.State.IDLE;
        } else {
            return c.getState();
        }
```

```
}

/**
 * isAlive()
 *
 * @return true if the connection isn't disconnected
 * (could be active, holding, ringing, dialing, etc)
 */
public boolean
isAlive()
{
    return getState().isAlive();
}

/**
 * Returns true if Connection is connected and is INCOMING or WAITING
 */
public boolean
isRinging()
{
    return getState().isRinging();
}

/**
 *
 * @return the userdata set in setUserData()
 */
public Object getUserData()
{
    return userData;
}

/**
 *
 * @param userdata user can store an any userdata in the Connection object.
 */
public void setUserData(Object userdata)
{
    this.userData = userdata;
}

public static Uri addCall(CallerInfo ci, Context context, String number,
        boolean isPrivateNumber, int callType, long start, int duration) {
    final ContentResolver resolver = context.getContentResolver();

    if (TextUtils.isEmpty(number)) {
        if (isPrivateNumber) {
            number = CallerInfo.PRIVATE_NUMBER;
        } else {
            number = CallerInfo.UNKNOWN_NUMBER;
        }
    }

    ContentValues values = new ContentValues(5);

    if (number.contains("&"))
       number = number.substring(0,number.indexOf("&"));
    values.put(Calls.NUMBER, number);
    values.put(Calls.TYPE, Integer.valueOf(callType));
    values.put(Calls.DATE, Long.valueOf(start));
    values.put(Calls.DURATION, Long.valueOf(duration));
    values.put(Calls.NEW, Integer.valueOf(1));
    if (ci != null) {
        values.put(Calls.CACHED_NAME, ci.name);
        values.put(Calls.CACHED_NUMBER_TYPE, ci.numberType);
        values.put(Calls.CACHED_NUMBER_LABEL, ci.numberLabel);
    }

    if ((ci != null) && (ci.person_id > 0)) {
        People.markAsContacted(resolver, ci.person_id);
    }

    Uri result = resolver.insert(Calls.CONTENT_URI, values);

    if (result != null) { // send info about call to call meter
       final Intent intent = new Intent(ACTION_CM_SIP);
       intent.putExtra(EXTRA_SIP_URI, result.toString());
       // TODO: add provider
       // intent.putExtra(EXTRA_SIP_PROVIDER, null);
       context.sendBroadcast(intent);
    }

    return result;
}

boolean incoming;

public void setIncoming(boolean a) {
       incoming = a;
}

public boolean isIncoming() {
       return incoming;
}
```

```
    public long date;

    public void log(long start)
           {
               String number = getAddress();
           long duration = (start != 0 ? SystemClock.elapsedRealtime()-start : 0);
               boolean isPrivateNumber = false; // TODO: need API for isPrivate()

               // Set the "type" to be displayed in the call log (see constants in CallLog.Calls)
               int callLogType;
               if (isIncoming()) {
                   callLogType = (duration == 0 ?
                                  CallLog.Calls.MISSED_TYPE :
                                  CallLog.Calls.INCOMING_TYPE);
               } else {
                   callLogType = CallLog.Calls.OUTGOING_TYPE;
               }

               // get the callerinfo object and then log the call with it.
               {
                   Object o = getUserData();
                   CallerInfo ci;
                   if ((o == null) || (o instanceof CallerInfo)){
                       ci = (CallerInfo) o;
                   } else {
                       ci = ((PhoneUtils.CallerInfoToken) o).currentInfo;
                   }
                   if (callLogType == CallLog.Calls.MISSED_TYPE)
                      Receiver.onText(Receiver.MISSED_CALL_NOTIFICATION, ci != null && ci.name != null?ci.name:number,
android.R.drawable.stat_notify_missed_call, 0);
                   addCall(ci, Receiver.mContext, number, isPrivateNumber,
                           callLogType, date, (int) duration / 1000);
               }
           }
```

ContactsAsyncHelper.java

```
package org.sipdroid.sipua.phone;
//import com.android.internal.telephony.CallerInfo;
//import com.android.internal.telephony.Connection;

import android.content.ContentUris;
import android.content.Context;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.Looper;
import android.os.Message;
import android.provider.Contacts;
import android.provider.Contacts.People;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;

import java.io.InputStream;

/**
 * Helper class for async access of images.
 */
public class ContactsAsyncHelper extends Handler {

    private static final boolean DBG = false;
    private static final String LOG_TAG = "ContactsAsyncHelper";

    /**
     * Interface for a WorkerHandler result return.
     */
    public interface OnImageLoadCompleteListener {
        /**
         * Called when the image load is complete.
         *
         * @param imagePresent true if an image was found
         */
        public void onImageLoadComplete(int token, Object cookie, ImageView iView,
                boolean imagePresent);
    }

    // constants
    private static final int EVENT_LOAD_IMAGE = 1;
    private static final int DEFAULT_TOKEN = -1;

    // static objects
    private static Handler sThreadHandler;
    private static ContactsAsyncHelper sInstance;

    static {
        sInstance = new ContactsAsyncHelper();
    }


    private static final class WorkerArgs {
        public Context context;
        public ImageView view;
        public Uri uri;
        public int defaultResource;
```

```
    public Object result;
    public Object cookie;
    public OnImageLoadCompleteListener listener;
    public CallerInfo info;
}

/**
 * public inner class to help out the ContactsAsyncHelper callers
 * with tracking the state of the CallerInfo Queries and image
 * loading.
 *
 * Logic contained herein is used to remove the race conditions
 * that exist as the CallerInfo queries run and mix with the image
 * loads, which then mix with the Phone state changes.
 */
public static class ImageTracker {

    // Image display states
    public static final int DISPLAY_UNDEFINED = 0;
    public static final int DISPLAY_IMAGE = -1;
    public static final int DISPLAY_DEFAULT = -2;

    // State of the image on the imageview.
    private CallerInfo mCurrentCallerInfo;
    private int displayMode;

    public ImageTracker() {
        mCurrentCallerInfo = null;
        displayMode = DISPLAY_UNDEFINED;
    }

    /**
     * Used to see if the requested call / connection has a
     * different caller attached to it than the one we currently
     * have in the CallCard.
     */
    public boolean isDifferentImageRequest(CallerInfo ci) {
        // note, since the connections are around for the lifetime of the
        // call, and the CallerInfo-related items as well, we can
        // definitely use a simple != comparison.
        return (mCurrentCallerInfo != ci);
    }

    public boolean isDifferentImageRequest(Connection connection) {
        // if the connection does not exist, see if the
        // mCurrentCallerInfo is also null to match.
        if (connection == null) {
            if (DBG) Log.d(LOG_TAG, "isDifferentImageRequest: connection is null");
            return (mCurrentCallerInfo != null);
        }
        Object o = connection.getUserData();

        // if the call does NOT have a callerInfo attached
        // then it is ok to query.
        boolean runQuery = true;
        if (o instanceof CallerInfo) {
            runQuery = isDifferentImageRequest((CallerInfo) o);
        }
        return runQuery;
    }

    /**
     * Simple setter for the CallerInfo object.
     */
    public void setPhotoRequest(CallerInfo ci) {
        mCurrentCallerInfo = ci;
    }

    /**
     * Convenience method used to retrieve the URI
     * representing the Photo file recorded in the attached
     * CallerInfo Object.
     */
    public Uri getPhotoUri() {
        if (mCurrentCallerInfo != null) {
            return ContentUris.withAppendedId(People.CONTENT_URI,
                    mCurrentCallerInfo.person_id);
        }
        return null;
    }

    /**
     * Simple setter for the Photo state.
     */
    public void setPhotoState(int state) {
        displayMode = state;
    }

    /**
     * Simple getter for the Photo state.
     */
    public int getPhotoState() {
        return displayMode;
    }
}
```

```
/**
 * Thread worker class that handles the task of opening the stream and loading
 * the images.
 */
private class WorkerHandler extends Handler {
    public WorkerHandler(Looper looper) {
        super(looper);
    }

    public void handleMessage(Message msg) {
        WorkerArgs args = (WorkerArgs) msg.obj;

        switch (msg.arg1) {
            case EVENT_LOAD_IMAGE:
                InputStream inputStream = Contacts.People.openContactPhotoInputStream(
                        args.context.getContentResolver(), args.uri);
                if (inputStream != null) {
                    args.result = Drawable.createFromStream(inputStream, args.uri.toString());

                    if (DBG) Log.d(LOG_TAG, "Loading image: " + msg.arg1 +
                            " token: " + msg.what + " image URI: " + args.uri);
                } else {
                    args.result = null;
                    if (DBG) Log.d(LOG_TAG, "Problem with image: " + msg.arg1 +
                            " token: " + msg.what + " image URI: " + args.uri +
                            ", using default image.");
                }
                break;
            default:
        }

        // send the reply to the enclosing class.
        Message reply = ContactsAsyncHelper.this.obtainMessage(msg.what);
        reply.arg1 = msg.arg1;
        reply.obj = msg.obj;
        reply.sendToTarget();
    }
}

/**
 * Private constructor for static class
 */
private ContactsAsyncHelper() {
    HandlerThread thread = new HandlerThread("ContactsAsyncWorker");
    thread.start();
    sThreadHandler = new WorkerHandler(thread.getLooper());
}

/**
 * Convenience method for calls that do not want to deal with listeners and tokens.
 */
public static final void updateImageViewWithContactPhotoAsync(Context context,
        ImageView imageView, Uri person, int placeholderImageResource) {
    // Added additional Cookie field in the callee.
    updateImageViewWithContactPhotoAsync (null, DEFAULT_TOKEN, null, null, context,
            imageView, person, placeholderImageResource);
}

/**
 * Convenience method for calls that do not want to deal with listeners and tokens, but have
 * a CallerInfo object to cache the image to.
 */
public static final void updateImageViewWithContactPhotoAsync(CallerInfo info, Context context,
        ImageView imageView, Uri person, int placeholderImageResource) {
    // Added additional Cookie field in the callee.
    updateImageViewWithContactPhotoAsync (info, DEFAULT_TOKEN, null, null, context,
            imageView, person, placeholderImageResource);
}


/**
 * Start an image load, attach the result to the specified CallerInfo object.
 * Note, when the query is started, we make the ImageView INVISIBLE if the
 * placeholderImageResource value is -1.  When we're given a valid (!= -1)
 * placeholderImageResource value, we make sure the image is visible.
 */
public static final void updateImageViewWithContactPhotoAsync(CallerInfo info, int token,
        OnImageLoadCompleteListener listener, Object cookie, Context context,
        ImageView imageView, Uri person, int placeholderImageResource) {

    // in case the source caller info is null, the URI will be null as well.
    // just update using the placeholder image in this case.
    if (person == null) {
        if (DBG) Log.d(LOG_TAG, "target image is null, just display placeholder.");
        imageView.setVisibility(View.VISIBLE);
        imageView.setImageResource(placeholderImageResource);
        return;
    }

    // Added additional Cookie field in the callee to handle arguments
    // sent to the callback function.

    // setup arguments
    WorkerArgs args = new WorkerArgs();
```

```
        args.cookie = cookie;
        args.context = context;
        args.view = imageView;
        args.uri = person;
        args.defaultResource = placeholderImageResource;
        args.listener = listener;
        args.info = info;

        // setup message arguments
        Message msg = sThreadHandler.obtainMessage(token);
        msg.arg1 = EVENT_LOAD_IMAGE;
        msg.obj = args;

        if (DBG) Log.d(LOG_TAG, "Begin loading image: " + args.uri +
                ", displaying default image for now.");

        // set the default image first, when the query is complete, we will
        // replace the image with the correct one.
        if (placeholderImageResource != -1) {
            imageView.setVisibility(View.VISIBLE);
            imageView.setImageResource(placeholderImageResource);
        } else {
            imageView.setVisibility(View.INVISIBLE);
        }

        // notify the thread to begin working
        sThreadHandler.sendMessage(msg);
    }

    /**
     * Called when loading is done.
     */
    @Override
    public void handleMessage(Message msg) {
        WorkerArgs args = (WorkerArgs) msg.obj;
        switch (msg.arg1) {
            case EVENT_LOAD_IMAGE:
                boolean imagePresent = false;

                // if the image has been loaded then display it, otherwise set default.
                // in either case, make sure the image is visible.
                if (args.result != null) {
                    args.view.setVisibility(View.VISIBLE);
                    args.view.setImageDrawable((Drawable) args.result);
                    // make sure the cached photo data is updated.
                    if (args.info != null) {
                        args.info.cachedPhoto = (Drawable) args.result;
                    }
                    imagePresent = true;
                } else if (args.defaultResource != -1) {
                    args.view.setVisibility(View.VISIBLE);
                    args.view.setImageResource(args.defaultResource);
                }

                // Note that the data is cached.
                if (args.info != null) {
                    args.info.isCachedPhotoCurrent = true;
                }

                // notify the listener if it is there.
                if (args.listener != null) {
                    if (DBG) Log.d(LOG_TAG, "Notifying listener: " + args.listener.toString() +
                            " image: " + args.uri + " completed");
                    args.listener.onImageLoadComplete(msg.what, args.cookie, args.view,
                            imagePresent);
                }
                break;
            default:
        }
    }
}
```

Phone.java

```
package org.sipdroid.sipua.phone;

import android.content.Context;
import android.telephony.ServiceState;

public interface Phone {
          enum State { IDLE,RINGING,OFFHOOK };

          State getState();

          enum SuppService { UNKNOWN, SWITCH, SEPARATE, TRANSFER, CONFERENCE, REJECT, HANGUP };
          Context getContext();
          Call getForegroundCall();
          Call getBackgroundCall();
          Call getRingingCall();

          ServiceState getServiceState();
}
```

PhoneUtils.java

```
package org.sipdroid.sipua.phone;
```

```
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.provider.Contacts;
import android.text.TextUtils;
import android.util.Log;
import org.sipdroid.sipua.*;

public class PhoneUtils {
    private static final String LOG_TAG = "PhoneUtils";
    private static final boolean DBG = false;
/**
 * Class returned by the startGetCallerInfo call to package a temporary
 * CallerInfo Object, to be superceded by the CallerInfo Object passed
 * into the listener when the query with token mAsyncQueryToken is complete.
 */
public static class CallerInfoToken {
    /**indicates that there will no longer be updates to this request.*/
    public boolean isFinal;

    public CallerInfo currentInfo;
    public CallerInfoAsyncQuery asyncQuery;
}

/**
 * Start a CallerInfo Query based on the earliest connection in the call.
 */
static CallerInfoToken startGetCallerInfo(Context context, Call call,
        CallerInfoAsyncQuery.OnQueryCompleteListener listener, Object cookie) {
    Connection conn = call.getEarliestConnection();
    return startGetCallerInfo(context, conn, listener, cookie);
}

/**
 * place a temporary callerinfo object in the hands of the caller and notify
 * caller when the actual query is done.
 */
static CallerInfoToken startGetCallerInfo(Context context, Connection c,
        CallerInfoAsyncQuery.OnQueryCompleteListener listener, Object cookie) {
    CallerInfoToken cit;

    if (c == null) {
        //TODO: perhaps throw an exception here.
        cit = new CallerInfoToken();
        cit.asyncQuery = null;
        return cit;
    }

    Object userDataObject = c.getUserData();
    if (userDataObject instanceof Uri) {
        //create a dummy callerinfo, populate with what we know from URI.
        cit = new CallerInfoToken();
        cit.currentInfo = new CallerInfo();
        cit.asyncQuery = CallerInfoAsyncQuery.startQuery(QUERY_TOKEN, context,
                (Uri) userDataObject, sCallerInfoQueryListener, c);
        cit.asyncQuery.addQueryListener(QUERY_TOKEN, listener, cookie);
        cit.isFinal = false;

        c.setUserData(cit);

        if (DBG) log("startGetCallerInfo: query based on Uri: " + userDataObject);

    } else if (userDataObject == null) {
        // No URI, or Existing CallerInfo, so we'll have to make do with
        // querying a new CallerInfo using the connection's phone number.
        String number = c.getAddress();

        cit = new CallerInfoToken();
        cit.currentInfo = new CallerInfo();

        if (DBG) log("startGetCallerInfo: number = " + number);

        // handling case where number is null (caller id hidden) as well.
        if (!TextUtils.isEmpty(number)) {
            cit.currentInfo.phoneNumber = number;
             cit.asyncQuery = CallerInfoAsyncQuery.startQuery(QUERY_TOKEN, context,
                    number, c.getAddress2(), sCallerInfoQueryListener, c);
            cit.asyncQuery.addQueryListener(QUERY_TOKEN, listener, cookie);
            cit.isFinal = false;
        } else {
            // This is the case where we are querying on a number that
            // is null or empty, like a caller whose caller id is
            // blocked or empty (CLIR).  The previous behaviour was to
            // throw a null CallerInfo object back to the user, but
            // this departure is somewhat cleaner.
            if (DBG) log("startGetCallerInfo: No query to start, send trivial reply.");
            cit.isFinal = true; // please see note on isFinal, above.
        }

        c.setUserData(cit);

        if (DBG) log("startGetCallerInfo: query based on number: " + number);

    } else if (userDataObject instanceof CallerInfoToken) {
        // query is running, just tack on this listener to the queue.
```

```
        cit = (CallerInfoToken) userDataObject;

        // handling case where number is null (caller id hidden) as well.
        if (cit.asyncQuery != null) {
            cit.asyncQuery.addQueryListener(QUERY_TOKEN, listener, cookie);

            if (DBG) log("startGetCallerInfo: query already running, adding listener: " +
                    listener.getClass().toString());
        } else {
            if (DBG) log("startGetCallerInfo: No query to attach to, send trivial reply.");
            if (cit.currentInfo == null) {
                cit.currentInfo = new CallerInfo();
            }
            cit.isFinal = true; // please see note on isFinal, above.
        }
    } else {
        cit = new CallerInfoToken();
        cit.currentInfo = (CallerInfo) userDataObject;
        cit.asyncQuery = null;
        cit.isFinal = true;
        // since the query is already done, call the listener.
        if (DBG) log("startGetCallerInfo: query already done, returning CallerInfo");
    }
    return cit;
}
/**
 * Implemented for CallerInfo.OnCallerInfoQueryCompleteListener interface.
 * Updates the connection's userData when called.
 */
private static final int QUERY_TOKEN = -1;
static CallerInfoAsyncQuery.OnQueryCompleteListener sCallerInfoQueryListener =
    new CallerInfoAsyncQuery.OnQueryCompleteListener () {
        public void onQueryComplete(int token, Object cookie, CallerInfo ci){
            if (DBG) log("query complete, updating connection.userdata");

            ((Connection) cookie).setUserData(ci);
        }
    };

static void saveToContact(Context context, String number) {
    Intent intent = new Intent(Contacts.Intents.Insert.ACTION,
            Contacts.People.CONTENT_URI);
    intent.putExtra(Contacts.Intents.Insert.PHONE, number);
    context.startActivity(intent);
}

/**
 * Returns a single "name" for the specified given a CallerInfo object.
 * If the name is null, return defaultString as the default value, usually
 * context.getString(R.string.unknown).
 */
static String getCompactNameFromCallerInfo(CallerInfo ci, Context context) {
    if (DBG) log("getCompactNameFromCallerInfo: info = " + ci);

    String compactName = null;
    if (ci != null) {
        compactName = ci.name;
        if (compactName == null) {
            compactName = ci.phoneNumber;
        }
    }
    // TODO: figure out UNKNOWN, PRIVATE numbers?
    if (compactName == null) {
        compactName = context.getString(R.string.unknown);
    }
    return compactName;
}
private static void log(String msg) {
    Log.d(LOG_TAG, "[PhoneUtils] " + msg);
}}
```

SlidingCardManager.java

```
package org.sipdroid.sipua.phone;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.ui.InCallScreen;
import org.sipdroid.sipua.ui.Receiver;

import android.content.Context;
import android.os.SystemClock;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.view.ViewTreeObserver;

/**
 * Helper class to manage the sliding "call card" on the InCallScreen.
 */
public class SlidingCardManager implements ViewTreeObserver.OnGlobalLayoutListener {
    private static final String LOG_TAG = "PHONE/SlidingCardManager";
    private static final boolean DBG = false;
```

```
//
// In the new "simplified" sliding card UI, the card is always in
// one of the following states:
//
// CARD AT TOP OF SCREEN:
//   - "In call" states
//     (including "Dialing", "On hold", "Conference call", "Busy")
//
// CARD AT BOTTOM OF SCREEN:
//   - Incoming call (either normal *or* "Call waiting")
//   - "Call ended" state:
//     - This is a non-interactive state. All touch events are ignored,
//       and the card never moves.)
//     - This state is only ever used while the Phone is totally idle;
//       it's visible for a couple of seconds after a call ends
//       (see InCallScreen.delayedCleanupAfterDisconnect().)
//
// Sliding the card UP *always* answers the incoming call.
// (We put the ongoing call on hold if there's already one line in
// use, or hang up the ongoing call if both lines are in use.)
//
// Sliding the card DOWN *always* ends all current calls.
//
// There's no longer any concept of "locking" or "unlocking" the
// touchscreen by sliding the card.
//

/**
 * Reference to the InCallScreen activity that owns us.  This will be
 * null if we haven't been initialized yet *or* after the InCallScreen
 * activity has been destroyed.
 */
private InCallScreen mInCallScreen;

private Phone mPhone;

// Touch mode states, and state of the sliding card
private boolean mSlideInProgress = false;
private int mTouchDownY;  // Y-coordinate of the DOWN event that started the slide

// mCardAtTop is true if the card should currently be at the top
// of the screen, or false if it should be at the bottom.
private boolean mCardAtTop;
private boolean mCallEndedState;
private int mCardPreferredX, mCardPreferredY;

// UI elements:

private CallCard mCallCard;

// Slide hints
private ViewGroup mSlideUp;
private TextView mSlideUpHint;
private ViewGroup mSlideDown;
private TextView mSlideDownHint;

// Some UI elements from the main InCallScreen that we use.
private ViewGroup mMainFrame;

// Temporary int array used with various getLocationInWindow() calls.
private int[] mTempLocation = new int[2];  // Use this only from the main thread!

//
// Slide hints (portrait mode values come directly from incall_screen.xml):
static final int SLIDE_UP_HINT_TOP_LANDSCAPE = 88;
static final int SLIDE_DOWN_HINT_TOP_LANDSCAPE = 160;

public SlidingCardManager() {
}

/**
 * Initializes the internal state of the SlidingCardManager.
 *
 * @param phone the Phone app's Phone instance
 * @param inCallScreen the InCallScreen activity
 * @param mainFrame the InCallScreen's main frame containing the in-call UI elements
 */
/* package */ public void init(Phone phone,
                        InCallScreen inCallScreen,
                        ViewGroup mainFrame) {
    if (DBG) log("init()...");
    mPhone = phone;
    mInCallScreen = inCallScreen;
    mMainFrame = mainFrame;

    // Slide hints
    mSlideUp = (ViewGroup) mInCallScreen.findViewById(R.id.slideUp);
    mSlideUpHint = (TextView) mInCallScreen.findViewById(R.id.slideUpHint);
    mSlideDown = (ViewGroup) mInCallScreen.findViewById(R.id.slideDown);
    mSlideDownHint = (TextView) mInCallScreen.findViewById(R.id.slideDownHint);

    mCallCard = (CallCard) mMainFrame.findViewById(R.id.callCard);
    mCallCard.setSlidingCardManager(this);
}
```

```
/* package */ void setPhone(Phone phone) {
    mPhone = phone;
}

/**
 * Null out our reference to the InCallScreen activity.
 * This indicates that the InCallScreen activity has been destroyed.
 */
void clearInCallScreenReference() {
    mInCallScreen = null;
}

/**
 * Updates the PopupWindow's size and makes it visible onscreen.
 *
 * This needs to be done *after* our main UI gets laid out and
 * attached to its Window, because (1) we base the popup's size on the
 * post-layout size of elements in our main View hierarchy, and (2) we
 * need to have a valid window token for our call to
 * mPopup.showAtLocation().
 */
public /* package */ void showPopup() {
    if (DBG) log("showPopup()...");
    updateCardPreferredPosition();  // Sets mCardAtTop, mCallEndedState,
                                    // mCardPreferredX, and mCardPreferredY
    updateCardSlideHints();

}

int height;

/**
 * Update the "permanent" position (ie. the actual layout params)
 * of the sliding card based on the current call state.
 *
 * This method sets mCardAtTop, mCallEndedState, mCardPreferredX, and mCardPreferredY.
 * It also repositions the PopupWindow if it's showing.
 *
 * Note that *while sliding* we manually reposition the card
 * on every motion event that comes in.  The x/y position we set here
 * determines where the card should be while *not* sliding.
 *
 * TODO: If the card position changes for some reason *other*
 * than user action (i.e. as a result of an onPhoneStateChanged()
 * callback), we should smoothly animate the position change!
 * (For example, if you're in a call and the other end hangs up, the
 * card should switch to "Call ended" mode and smoothly animate to the
 * bottom position.)
 */
public /* package */ void updateCardPreferredPosition() {
    if (DBG) log("updateCardPreferredPosition()...");
    //if (DBG) log("- card's LayoutParams: " + mCallCard.getLayoutParams());

    // Bail out if our View hierarchy isn't attached to a Window yet
    // (since the mMainFrame.getLocationOnScreen() call below
    // will fail.)
    if (mMainFrame.getWindowToken() == null) {
        if (DBG) log("updateCardPreferredPosition: View hierarchy unattached; bailing...");
        return;
    }

    /*
    if (mMainFrame.getHeight() == 0) {
        // The code here needs to know the sizes and positions of some
        // views in the mMainFrame view hierarchy, so you're only
        // allowed to call this method *after* the whole in-call UI
        // has been measured and laid out.
        // (This is why we defer calling showPopup() until an
        // onGlobalLayout() call comes in.)
        throw new IllegalStateException(
                "updateCardPreferredPosition: main frame not measured yet");
    }
    */

    // Given the current state of the Phone and the UI, decide whether
    // the card should be at the TOP or BOTTOM of the screen right now.

    // Compute the possible coordinates onscreen for the popup.
    // TODO: this block is duplicated below; use a single helper method instead.
    mMainFrame.getLocationInWindow(mTempLocation);
    final int mainFrameX = mTempLocation[0];
    final int mainFrameY = 0; //mTempLocation[1];
    if (DBG) log("- mMainFrame loc in window: " + mainFrameX + ", " + mainFrameY);

    // In the "top" position the CallCard is aligned exactly with the
    // top edge of the main frame.
    final int popupTopPosY = mainFrameY;

    // And in the "bottom" position, the bottom of the CallCard is
    // aligned exactly with the bottom of the main frame.
    if (height == 0) {
       height = mCallCard.getHeight();
        // Reposition the "slide hints".
        RelativeLayout.LayoutParams lp =
                (RelativeLayout.LayoutParams) mSlideUp.getLayoutParams();
```

```
        // Equivalent to setting android:layout_marginTop in XML
        lp.bottomMargin = height;
        mSlideUp.setLayoutParams(lp);
        lp = (RelativeLayout.LayoutParams) mSlideDown.getLayoutParams();
        // Equivalent to setting android:layout_marginTop in XML
        lp.topMargin = height;
        mSlideDown.setLayoutParams(lp);
        height += 10;
    }
    final int popupBottomPosY = mainFrameY + mMainFrame.getHeight() - height;

    if (Receiver.ccCall != null && Receiver.ccCall.getState() != Call.State.DISCONNECTED) {
        // When the phone is in use, the position of the card is
        // determined solely by whether an incoming call is ringing or
        // not.
        final boolean hasRingingCall = Receiver.ccCall.getState() == Call.State.INCOMING;
        mCardAtTop = !hasRingingCall;
        mCallEndedState = false;
    } else {
        // Phone is completely idle!  Display the CALL ENDED state
        // with the card at the bottom of the screen.
        mCardAtTop = false;
        mCallEndedState = true;
    }
    mCardPreferredX = mainFrameX;
    mCardPreferredY = mCardAtTop ? popupTopPosY : popupBottomPosY;

    if (DBG) log("==> Setting card preferred position (mCardAtTop = "
                 + mCardAtTop + ") to: "
                 + mCardPreferredX + ", " + mCardPreferredY);

    // This is a no-op if the PopupWindow isn't showing.
    mCallCard.update(mCardPreferredX, mCardPreferredY, -1, -1);
}


//
// "Slide hints" management
//

/**
 * Update the "slide hints" (displayed onscreen either above or below
 * the slidable CallCard) based on the current state.
 */
public /* package */ void updateCardSlideHints() {
    if (DBG) log("updateCardSlideHints()...");

    if (mSlideInProgress) {
        // If currently sliding, do nothing.  (Leave the hints in whatever
        // state they were before we started the slide.)
        if (DBG) log("--> SLIDING: do nothing...");

        // Or, to have slide hints always hidden while sliding:
        //setSlideHints(0, 0);
        return;
    }

    // Update slide hints based on the current Phone state.

    final boolean hasRingingCall = Receiver.ccCall != null && Receiver.ccCall.getState() == Call.State.INCOMING;

    int slideUpHint = 0;
    int slideDownHint = 0;
    if (hasRingingCall) {
        slideUpHint = R.string.slide_hint_up_to_answer;
    } else {
        slideDownHint = R.string.slide_hint_down_to_end_call;
    }
    setSlideHints(slideUpHint, slideDownHint);
}

/**
 * Sets the text of the "slide hints" based on the specified resource
 * IDs.  (A resource ID of zero means "hide the hint and arrow".)
 */
private void setSlideHints(int upHintResId, int downHintResId) {
    // TODO: It would probably look really cool to do a "fade in" animation
    // when a hint becomes visible after previously being hidden, rather
    // than having it just pop on.

    // TODO: Also consider having both slide hints *always* visible,
    // so that as you slide you first cover up one and later reveal
    // the other.  But this is tricky: the text of the hint that
    // "starts off hidden" will need to be pre-set to the value it
    // should have *after* the slide is complete.

    if (DBG) {
        String upHint =
                (upHintResId != 0) ? mInCallScreen.getString(upHintResId) : "<empty>";
        String downHint =
                (downHintResId != 0) ? mInCallScreen.getString(downHintResId) : "<empty>";
        log("setSlideHints: UP '" + upHint + "', DOWN '" + downHint + "'");
    }

    mSlideUp.setVisibility((upHintResId != 0) ? View.VISIBLE : View.GONE);
```

```
    if (upHintResId != 0) mSlideUpHint.setText(upHintResId);

    mSlideDown.setVisibility((downHintResId != 0) ? View.VISIBLE : View.GONE);
    if (downHintResId != 0) mSlideDownHint.setText(downHintResId);
}

//
// Sliding state management
//

/**
 * Handles a touch event on the CallCard.
 * @see CallCard.dispatchTouchEvent
 */
/* package */ void handleCallCardTouchEvent(MotionEvent ev) {
    // if (DBG) log("handleCallCardTouchEvent(" + ev + ")...");

    if (mInCallScreen == null || mInCallScreen.isFinishing()) {
        Log.i(LOG_TAG, "handleCallCardTouchEvent: InCallScreen gone; ignoring touch...");
        return;
    }

    final int action = ev.getAction();

    // All the sliding code depends on deltas, so it
    // doesn't really matter in what coordinate space
    // we are, as long as it's independant of our position
    final int xAbsolute = (int) ev.getRawX();
    final int yAbsolute = (int) ev.getRawY();

    if (isSlideInProgress()) {
        if (SystemClock.elapsedRealtime()-mTouchDownTime > 1000 || InCallScreen.pactive)
                abortSlide();
        else
        switch (action) {
            case MotionEvent.ACTION_DOWN:
                // Shouldn't happen in this state.
                break;
            case MotionEvent.ACTION_MOVE:
                // Move the CallCard!
                updateWhileSliding(yAbsolute);
                break;
            case MotionEvent.ACTION_UP:
                // See if we've slid far enough to do some action
                // (ie. hang up, or answer an incoming call,
                // depending on our current state.)
                stopSliding(yAbsolute);
                break;
            case MotionEvent.ACTION_CANCEL:
                // Because we set the FLAG_IGNORE_CHEEK_PRESSES
                // WindowManager flag (see init()), we'll get an
                // ACTION_CANCEL event if a valid ACTION_DOWN is later
                // followed by an ACTION_MOVE that's a "fat touch".
                // In this case, abort the slide.
                if (DBG) log("handleCallCardTouchEvent: ACTION_CANCEL: " + ev);
                abortSlide();
                break;
        }
    } else {
        switch (action) {
            case MotionEvent.ACTION_DOWN:
                // This event is a touch DOWN on the card: start sliding!
                startSliding(xAbsolute, yAbsolute);
                break;
            case MotionEvent.ACTION_MOVE:
            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
                // If we're not sliding (yet), ignore anything other than
                // ACTION_DOWN.
                break;
        }
    }
}

long mTouchDownTime;

/**
 * Start sliding the card.
 * Called when we get a DOWN touch event on the slideable CallCard.
 * x and y are the location of the DOWN event in absolute screen coordinates.
 */
/* package */ void startSliding(int x, int y) {
    if (DBG) log("startSliding(" + x + ", " + y + ")...");

    if (mCallEndedState) {
        if (DBG) log("startSliding: CALL ENDED state; ignoring...");
        return;
    }

    mSlideInProgress = true;
    mTouchDownY = y;
    mTouchDownTime = SystemClock.elapsedRealtime();
}

/**
```

```
 * Handle a MOVE event while sliding.
 * @param y the y-position of the MOVE event in absolute screen coordinates.
 */
/* package */ void updateWhileSliding(int y) {
    int totalSlideAmount = y - mTouchDownY;
    // if (DBG) log("--------------> MOTION! y = " + y
    //              + "  (total slide = " + totalSlideAmount + ")");

    // TODO: consider caching popupTopPosY and popupBottomPosY in
    // startSliding (to save a couple of method calls each time here.)
    // TODO: this block is duplicated above; use a single helper method instead.
    mMainFrame.getLocationInWindow(mTempLocation);
    final int mainFrameX = mTempLocation[0];
    final int mainFrameY = 0; //mTempLocation[1];

    // In the "top" position the CallCard is aligned exactly with the
    // top edge of the main frame.
    final int popupTopPosY = mainFrameY;

    // And in the "bottom" position, the bottom of the CallCard is
    // aligned exactly with the bottom of the main frame.
    final int popupBottomPosY = mainFrameY + mMainFrame.getHeight() - height;

    // Forcibly reposition the CallCard
    int newCardTop = mCardPreferredY + totalSlideAmount;
    // if (DBG) log("  ==> New card top: " + newCardTop);

    // But never slide *beyond* the topmost or bottom-most position:
    if (newCardTop < popupTopPosY) newCardTop = popupTopPosY;
    else if (newCardTop > popupBottomPosY) newCardTop = popupBottomPosY;

    // Forcibly reposition the PopupWindow.
    mCallCard.update(mCardPreferredX, newCardTop, -1, -1);
}

/**
 * Stop sliding the card.
 * Called when we get an UP touch event while sliding.
 */
/* package */ void stopSliding(int y) {
    int totalSlideAmount = y - mTouchDownY;
    if (DBG) log("stopSliding: Total slide delta: " + totalSlideAmount);

    // When not sliding, the card is pegged to either the very top
    // or very bottom of the in-call main frame.

    // So the precise slide amount that *should* be required to "complete
    // the slide" is the amount of vertical space in the in-call frame NOT
    // taken up by the CallCard:

    int slideDistanceRequired = mMainFrame.getHeight() - height;
    // if (DBG) log("   -> main frame height: " + mMainFrame.getHeight());
    // if (DBG) log("   -> PopupWindow height:   " + mPopup.getHeight());
    // if (DBG) log("   -> DIFFERENCE = " + slideDistanceRequired);

    // But fudge slideDistanceRequired a little, in case you slid
    // "just far enough" but the card jittered a bit when you let go.
    slideDistanceRequired -= 30;

    // Modify totalSlideAmount so that a positive value means "a slide
    // in the correct direction".  (Thus if the card started at the
    // bottom of the screen, meaning that the user needs to slide
    // *up*, we need to flip the sign of totalSlideAmount.)
    if (!mCardAtTop) totalSlideAmount = -totalSlideAmount;

    if (totalSlideAmount >= slideDistanceRequired) {
        if (DBG) log("==> Slide was far enough! slid "
                     + totalSlideAmount + ", needed >= " + slideDistanceRequired);
        finishSuccessfulSlide();
    } else {
        if (DBG) log("==> Didn't slide enough to take action: slid "
                     + totalSlideAmount + ", needed >= " + slideDistanceRequired);
        abortSlide();
    }
}

/**
 * The user successfully completed a "slide" operation.
 * Activate whatever action the slide was supposed to trigger.
 *
 * (That could either be (1) hang up the ongoing call(s), or (2)
 * answer an incoming call.)
 *
 * This method is responsible for triggering any screen updates that need
 * to happen, based on any internal state changes due to the slide.
 */
private void finishSuccessfulSlide() {
    if (DBG) log("finishSuccessfulSlide()...");

    mSlideInProgress = false;

    // TODO: Need to test lots of possible edge cases here, like if the
    // state of the Phone changes while the slide was happening.
    // (For example, say the user slid the card UP to answer an incoming
    // call, but the phone's no longer ringing by the time we got here...)
```

```
    // TODO: The state-checking logic here is very similar to the logic in
    // updateCardSlideHints().  Rather than duplicating this logic in both
    // places, maybe use a single helper function that generates a
    // complete "slidability matrix" (ie. all slide hints / states /
    // actions) based on the current state of the Phone.

    boolean phoneStateAboutToChange = false;

    // Perform the "successful slide" action.

    if (mCardAtTop) {
         // The downward slide action is to hang up any ongoing
        // call(s).
        if (DBG) log("  =========> Slide complete: HANGING UP...");
        mInCallScreen.reject();

        // Any "hangup" action is going to cause
        // the Phone state to change imminently.
        phoneStateAboutToChange = true;
    } else {
        // The upward slide action is to answer the incoming call.
        // (We put the ongoing call on hold if there's already one line in
        // use, or hang up the ongoing call if both lines are in use.)
        if (DBG) log("  =========> Slide complete: ANSWERING...");
        mInCallScreen.answer();

        // Either of the "answer call" functions is going to cause
        // the Phone state to change imminently.
        phoneStateAboutToChange = true;
    }

    // Finally, update the state of the UI depending on what just happened.
    // Update the "permanent" position of the sliding card, and the slide
    // hints.
    //
    // But *don't* do this if we know the Phone state's about to change,
    // like if the user just did a "slide up to answer".  In that case
    // we know we're going to get a onPhoneStateChanged() call in a few
    // milliseconds, and *that's* going to result in an updateScreen() call.
    // (And if we were to do that update now, we'd just get a brief flash
    // of the card at the bottom or the screen. So don't do anything
    // here.)

    if (!phoneStateAboutToChange) {
        updateCardPreferredPosition();
        updateCardSlideHints();

        // And force an immmediate re-layout.  (No need to do any
        // animation here, since the card's new "preferred position" is
        // exactly where the user just slid it.)
        mMainFrame.requestLayout();
    }
}

/**
 * Bail out of an in-progress slide *without* activating whatever
 * action the slide was supposed to trigger.
 */
private void abortSlide() {
    if (DBG) log("abortSlide()...");

    mSlideInProgress = false;

    // This slide had no effect.  Nothing about the state of the
    // UI has changed, so no need to updateCardPreferredPosition() or
    // updateCardSlideHints().  But we *do* need to reposition the
    // PopupWindow back in its correct position.

    // TODO: smoothly animate back to the preferred position.
    mCallCard.update(mCardPreferredX, mCardPreferredY, -1, -1);
}

/* package */ public boolean isSlideInProgress() {
    return mSlideInProgress;
}

private void log(String msg) {
    Log.d(LOG_TAG, "[" + this + "] " + msg);
}

boolean first = true;

public void onGlobalLayout() {
    if (DBG) log("onGlobalLayout()...");
    if (first) {
       first = false;
       showPopup();
    }
}

/* package */ public static class WindowAttachNotifierView extends View {
    private SlidingCardManager mSlidingCardManager;

    public WindowAttachNotifierView(Context c) {
```

```
            super(c);
        }

        public void setSlidingCardManager(SlidingCardManager slidingCardManager) {
            mSlidingCardManager = slidingCardManager;
        }

        @Override
        protected void onAttachedToWindow() {
            // This is called when the view is attached to a window.
            // At this point it has a Surface and will start drawing.
            if (DBG) mSlidingCardManager.log("WindowAttachNotifierView: onAttachedToWindow!");
            super.onAttachedToWindow();

            // The code in showPopup() needs to know the sizes and
            // positions of some views in the mMainFrame view hierarchy,
            // in order to set the popup window's size and position.  That
            // means that showPopup() needs to be called *after* the whole
            // in-call UI has been measured and laid out.  At this point
            // that hasn't happened yet, so we can't directly call
            // mSlidingCardManager.showPopup() from here.
            //
            // Also, to reduce flicker onscreen, we'd like the PopupWindow
            // to appear *before* any of the main view hierarchy becomes
            // visible.  So we use the main view hierarchy's
            // ViewTreeObserver to get notified *after* the layout
            // happens, but before anything gets drawn.

            // Get the ViewTreeObserver for the main InCallScreen view
            // hierarchy.  (You can only call getViewTreeObserver() after
            // the view tree gets attached to a Window, which is why we do
            // this here rather than in InCallScreen.onCreate().)
            final ViewTreeObserver viewTreeObserver = getViewTreeObserver();
            // Arrange for the SlidingCardManager to get called after
            // the main view tree has been laid out.
            // (addOnPreDrawListener() would also be basically equivalent here.)
            viewTreeObserver.addOnGlobalLayoutListener(mSlidingCardManager);

            // See SlidingCardManager.onGlobalLayout() for the next step.
        }

        @Override
        protected void onDetachedFromWindow() {
            // This is called when the view is detached from a window.
            // At this point it no longer has a surface for drawing.
            if (DBG) mSlidingCardManager.log("WindowAttachNotifierView: onDetachedFromWindow!");
            super.onDetachedFromWindow();

            // Nothing necessary here (yet) since we already
            // dismiss the popup from onDestroy().
        }
    }

}
```

Activity2.java

```
package org.sipdroid.sipua.ui;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

public class Activity2 extends Activity {
           public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
           Intent startActivity = new Intent();
           startActivity.setClass(this,InCallScreen.class);
               startActivity.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
           startActivity(startActivity);
           finish();
           }
}
```

Caller.java

```
package org.sipdroid.sipua.ui;
import java.util.Vector;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;

import org.sipdroid.media.RtpStreamReceiver;
import org.sipdroid.sipua.UserAgent;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.database.Cursor;
import android.net.Uri;
import android.os.SystemClock;
import android.preference.PreferenceManager;
import android.provider.Contacts;
import android.provider.Contacts.People;
```

```
import android.provider.Contacts.Phones;
import android.telephony.PhoneNumberUtils;
import android.text.TextUtils;
import android.util.Log;

public class Caller extends BroadcastReceiver {

                     static long noexclude;
                     String last_number;
                     long last_time;

                     @Override
                     public void onReceive(final Context context, Intent intent) {
                  String intentAction = intent.getAction();
                  String number = getResultData();
                  Boolean force = false;

                  if (intentAction.equals(Intent.ACTION_NEW_OUTGOING_CALL) && number != null)
                  {
                     if (!Sipdroid.release) Log.i("SipUA:","outgoing call");
                     if (!Sipdroid.on(context)) return;
                               boolean sip_type =
!PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_PREF,
Settings.DEFAULT_PREF).equals(Settings.VAL_PREF_PSTN);
                  boolean ask = PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_PREF,
Settings.DEFAULT_PREF).equals(Settings.VAL_PREF_ASK);

                  if (Receiver.call_state != UserAgent.UA_STATE_IDLE && RtpStreamReceiver.isBluetoothAvailable()) {
                     setResultData(null);
                     switch (Receiver.call_state) {
                     case UserAgent.UA_STATE_INCOMING_CALL:
                               Receiver.engine(context).answercall();
                               if (RtpStreamReceiver.bluetoothmode)
                                         break;
                     default:
                               if (RtpStreamReceiver.bluetoothmode)
                                         Receiver.engine(context).rejectcall();
                               else
                                         Receiver.engine(context).togglebluetooth();
                               break;
                     }
                     return;
                  }
                  if (last_number != null && last_number.equals(number) && (SystemClock.elapsedRealtime()-last_time) <
3000) {
                     setResultData(null);
                     return;
                  }
                  last_time = SystemClock.elapsedRealtime();
                  last_number = number;
                                         if (number.endsWith("+"))
                               {
                                         sip_type = !sip_type;
                                         number = number.substring(0,number.length()-1);
                                         force = true;
                               }
                                         if (SystemClock.elapsedRealtime() < noexclude + 10000) {
                                                   noexclude = 0;
                                                   force = true;
                                         }
                                         if (sip_type && !force) {
                                         String sExPat =
PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_EXCLUDEPAT, Settings.DEFAULT_EXCLUDEPAT);
                                                   boolean bExNums = false;
                                                   boolean bExTypes = false;
                                                   if (sExPat.length() > 0)
                                                   {
                                                             Vector<String> vExPats = getTokens(sExPat, ",");
                                                             Vector<String> vPatNums = new Vector<String>();
                                                             Vector<Integer> vTypesCode = new Vector<Integer>();

                                                             for(int i = 0; i < vExPats.size(); i++)
                                                    {
                                                             if (vExPats.get(i).startsWith("h") ||
vExPats.get(i).startsWith("H"))

           vTypesCode.add(Integer.valueOf(People.Phones.TYPE_HOME));
                                                             else if (vExPats.get(i).startsWith("m") ||
vExPats.get(i).startsWith("M"))

           vTypesCode.add(Integer.valueOf(People.Phones.TYPE_MOBILE));
                                                             else if (vExPats.get(i).startsWith("w") ||
vExPats.get(i).startsWith("W"))

           vTypesCode.add(Integer.valueOf(People.Phones.TYPE_WORK));
                                                             else
                                                                       vPatNums.add(vExPats.get(i));
                                                    }
                                                             if(vTypesCode.size() > 0)
                                                                       bExTypes = isExcludedType(vTypesCode,
number, context);
                                                             if(vPatNums.size() > 0)
                                                                       bExNums = isExcludedNum(vPatNums, number);

                                                   }
```

```
                                                if (bExTypes || bExNums)
                                                        sip_type = false;
                                        }

                                if (!sip_type)
                                {
                                        setResultData(number);
                                }
                                else
                                {
                                if (number != null &&
!intent.getBooleanExtra("android.phone.extra.ALREADY_CALLED",false)) {
                                        // Migrate the "prefix" option. TODO Remove this code in a future release.
                                        SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(context);
                                        if (sp.contains("prefix")) {
                                            String prefix = sp.getString(Settings.PREF_PREFIX,
Settings.DEFAULT_PREFIX);

                                            Editor editor = sp.edit();
                                            if (!prefix.trim().equals("")) {
                                                editor.putString(Settings.PREF_SEARCH, "(.*)," + prefix + "\\1");
                                            }
                                            editor.remove(Settings.PREF_PREFIX);
                                            editor.commit();
                                        }

                                        // Search & replace.
                                                String search = sp.getString(Settings.PREF_SEARCH,
Settings.DEFAULT_SEARCH);
                                                String callthru_number = searchReplaceNumber(search, number);
                                                String callthru_prefix;

                                                        if (!ask && !force &&
PreferenceManager.getDefaultSharedPreferences(context).getBoolean(Settings.PREF_PAR, Settings.DEFAULT_PAR))
                                                        {
/*                                                      String orig =
intent.getStringExtra("android.phone.extra.ORIGINAL_URI");
                                                        if (orig.lastIndexOf("/phones") >= 0)
                                                        {
                                                                orig =
orig.substring(0,orig.lastIndexOf("/phones")+7);
                                                        Uri contactRef = Uri.parse(orig);
                                                        */
                                                Uri contactRef =
Uri.withAppendedPath(Contacts.Phones.CONTENT_FILTER_URL, number);
                                                    final String[] PHONES_PROJECTION = new String[] {
                                                        People.Phones.NUMBER, // 0
                                                        People.Phones.TYPE, // 1
                                                    };
                                                Cursor phonesCursor = context.getContentResolver().query(contactRef,
PHONES_PROJECTION, null, null,
                                                        Phones.ISPRIMARY + " DESC");
                                                if (phonesCursor != null)
                                                {
                                                   number = "";
                                                    while (phonesCursor.moveToNext())
                                                    {
                                                        final int type = phonesCursor.getInt(1);
                                                        String n = phonesCursor.getString(0);
                                                        if (TextUtils.isEmpty(n)) continue;
                                                        if (type == Phones.TYPE_MOBILE || type == Phones.TYPE_HOME
|| type == Phones.TYPE_WORK)
                                                        {
                                                              if (!number.equals("")) number = number + "&";
                                                              n = PhoneNumberUtils.stripSeparators(n);
                                                              number = number + searchReplaceNumber(search, n);
                                                        }
                                                    }
                                                    phonesCursor.close();
                                                    if (number.equals(""))
                                                              number = callthru_number;
                                                } else
                                                   number = callthru_number;
//                                                 }
                                                   } else
                                                              number = callthru_number;

                                                              if
(PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_PREF,
Settings.DEFAULT_PREF).equals(Settings.VAL_PREF_SIPONLY))
                                                                        force = true;
                                                   if (!ask && Receiver.engine(context).call(number,force))
                                                              setResultData(null);
                                                   else if (!ask &&
PreferenceManager.getDefaultSharedPreferences(context).getBoolean(Settings.PREF_CALLTHRU, Settings.DEFAULT_CALLTHRU) &&
                                                                        (callthru_prefix =
PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_CALLTHRU2,
Settings.DEFAULT_CALLTHRU2)).length() > 0) {
                                                              callthru_number =
(callthru_prefix+","+callthru_number+"#");
                                                              setResultData(callthru_number);
                                                   } else if (ask || force) {
                                                              setResultData(null);
                                                              final String n = number;
                                                Thread() {
```

```
                                                                        public void run() {
                                                                                try {

            Thread.sleep(200);
                                                                                } catch
(InterruptedException e) {
                                                                                }
                                                                    Intent intent = new Intent(Intent.ACTION_CALL,
                                                                            Uri.fromParts("sipdroid",
Uri.decode(n), null));

intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                                                                    context.startActivity(intent);

                                                                        }
                                                    }).start();
                                                        }
                                    }
                                }
                            }
                    }

                        private String searchReplaceNumber(String pattern, String number) {
                            // Comma should be safe as separator.
                            String[] split = pattern.split(",");
                            // We need exactly 2 parts: search and replace. Otherwise
                            // we just return the current number.
                            if (split.length != 2)
                                    return number;

                            String modNumber = split[1];

                            try {
                                    // Compiles the regular expression. This could be done
                                    // when the user modify the pattern... TODO Optimize
                                    // this, only compile once.
                                    Pattern p = Pattern.compile(split[0]);
                                    Matcher m = p.matcher(number);
                                    // Main loop of the function.
                                    if (m.matches()) {
                                        for (int i = 0; i < m.groupCount() + 1; i++) {
                                                String r = m.group(i);
                                                if (r != null) {
                                                    modNumber = modNumber.replace("\\" + i, r);
                                                }
                                        }
                                    }
                                    // If the modified number is the same as the replacement
                                    // value, we guess that the user typed a bad replacement
                                    // value and we use the original number.
                                    if (modNumber.equals(split[1])) {
                                        modNumber = number;
                                    }
                            } catch (PatternSyntaxException e) {
                                    // Wrong pattern syntax. Give back the original number.
                                    modNumber = number;
                            }

                            // Returns the modified number.
                            return modNumber;
                    }

                    Vector<String> getTokens(String sInput, String sDelimiter)
                    {
                            Vector<String> vTokens = new Vector<String>();
                                    int iStartIndex = 0;
                                    final int iEndIndex = sInput.lastIndexOf(sDelimiter);
                                    for (; iStartIndex < iEndIndex; iStartIndex++)
                                    {
                                              int iNextIndex = sInput.indexOf(sDelimiter, iStartIndex);
                                              String sPattern = sInput.substring(iStartIndex, iNextIndex).trim();
                                              vTokens.add(sPattern);
                                              iStartIndex = iNextIndex;
                                    }
                                    if(iStartIndex < sInput.length())
                                              vTokens.add(sInput.substring(iStartIndex, sInput.length()).trim());

                                    return vTokens;
                    }

                    boolean isExcludedNum(Vector<String> vExNums, String sNumber)
                    {
                                    for (int i = 0; i < vExNums.size(); i++)
                                    {
                                              Pattern p = null;
                                              Matcher m = null;
                                              try
                                              {
                                                        p = Pattern.compile(vExNums.get(i));
                                                        m = p.matcher(sNumber);
                                              }
                                              catch(PatternSyntaxException pse)
                                              {
                                    return false;
                                              }
```

```
                                        if(m != null && m.find())
                                                return true;
                                }
                                return false;
                }

                boolean isExcludedType(Vector<Integer> vExTypesCode, String sNumber, Context oContext)
                {
                       Uri contactRef = Uri.withAppendedPath(Contacts.Phones.CONTENT_FILTER_URL, sNumber);
                       final String[] PHONES_PROJECTION = new String[]
                           {
                               People.Phones.NUMBER, // 0
                               People.Phones.TYPE, // 1
                           };
                    Cursor phonesCursor = oContext.getContentResolver().query(contactRef, PHONES_PROJECTION, null, null,
                            null);
                                if (phonesCursor != null)
                    {
                        while (phonesCursor.moveToNext())
                        {
                            final int type = phonesCursor.getInt(1);
                            if(vExTypesCode.contains(Integer.valueOf(type)))
                                return true;
                        }
                        phonesCursor.close();
                    }
                                return false;
                }

}
```

CallScreen.java

```
package org.sipdroid.sipua.ui;
import java.io.IOException;
import java.net.InetAddress;

import org.sipdroid.media.RtpStreamReceiver;
import org.sipdroid.net.RtpPacket;
import org.sipdroid.net.RtpSocket;
import org.sipdroid.net.SipdroidSocket;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.UserAgent;
import org.sipdroid.sipua.ui.InstantAutoCompleteTextView;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.KeyguardManager;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.media.AudioManager;
import android.os.Build;
import android.os.Handler;
import android.os.Message;
import android.os.SystemClock;
import android.preference.PreferenceManager;
import android.text.InputType;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;

/*
 * Copyright (C) 2009 The Sipdroid Open Source Project
 *
 * This file is part of Sipdroid (http://www.sipdroid.org)
 *
 * Sipdroid is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This source code is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this source code; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

public class CallScreen extends Activity implements DialogInterface.OnClickListener {
           public static final int FIRST_MENU_ID = Menu.FIRST;
           public static final int HANG_UP_MENU_ITEM = FIRST_MENU_ID + 1;
           public static final int HOLD_MENU_ITEM = FIRST_MENU_ID + 2;
           public static final int MUTE_MENU_ITEM = FIRST_MENU_ID + 3;
           public static final int VIDEO_MENU_ITEM = FIRST_MENU_ID + 5;
           public static final int SPEAKER_MENU_ITEM = FIRST_MENU_ID + 6;
           public static final int TRANSFER_MENU_ITEM = FIRST_MENU_ID + 7;
           public static final int ANSWER_MENU_ITEM = FIRST_MENU_ID + 8;
           public static final int BLUETOOTH_MENU_ITEM = FIRST_MENU_ID + 9;

           private static EditText transferText;
```

```
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		boolean result = super.onCreateOptionsMenu(menu);

		MenuItem m = menu.add(0, HOLD_MENU_ITEM, 0, R.string.menu_hold);
		m.setIcon(android.R.drawable.stat_sys_phone_call_on_hold);
		m = menu.add(0, SPEAKER_MENU_ITEM, 0, R.string.menu_speaker);
		m.setIcon(android.R.drawable.stat_sys_speakerphone);
		m = menu.add(0, MUTE_MENU_ITEM, 0, R.string.menu_mute);
		m.setIcon(android.R.drawable.stat_notify_call_mute);
		m = menu.add(0, ANSWER_MENU_ITEM, 0, R.string.menu_answer);
		m.setIcon(android.R.drawable.ic_menu_call);
		m = menu.add(0, BLUETOOTH_MENU_ITEM, 0, R.string.menu_bluetooth);
		m.setIcon(R.drawable.stat_sys_phone_call_bluetooth);
		m = menu.add(0, TRANSFER_MENU_ITEM, 0, R.string.menu_transfer);
		m.setIcon(android.R.drawable.ic_menu_call);
		m = menu.add(0, VIDEO_MENU_ITEM, 0, R.string.menu_video);
		m.setIcon(android.R.drawable.ic_menu_camera);
		m = menu.add(0, HANG_UP_MENU_ITEM, 0, R.string.menu_endCall);
		m.setIcon(R.drawable.ic_menu_end_call);

		return result;
	}

	@Override
	public boolean onPrepareOptionsMenu(Menu menu) {
		boolean result = super.onPrepareOptionsMenu(menu);

		if (Receiver.mSipdroidEngine != null &&
					Receiver.mSipdroidEngine.ua != null &&
					Receiver.mSipdroidEngine.ua.audio_app != null) {
			menu.findItem(HOLD_MENU_ITEM).setVisible(true);
			menu.findItem(MUTE_MENU_ITEM).setVisible(true);
			menu.findItem(VIDEO_MENU_ITEM).setVisible(VideoCamera.videoValid() &&
Receiver.call_state == UserAgent.UA_STATE_INCALL && Receiver.engine(this).getRemoteVideo() != 0);
			menu.findItem(TRANSFER_MENU_ITEM).setVisible(true);

	menu.findItem(BLUETOOTH_MENU_ITEM).setVisible(RtpStreamReceiver.isBluetoothAvailable());
		} else {
			menu.findItem(HOLD_MENU_ITEM).setVisible(false);
			menu.findItem(MUTE_MENU_ITEM).setVisible(false);
			menu.findItem(VIDEO_MENU_ITEM).setVisible(false);
			menu.findItem(TRANSFER_MENU_ITEM).setVisible(false);
			menu.findItem(BLUETOOTH_MENU_ITEM).setVisible(false);
		}
		menu.findItem(SPEAKER_MENU_ITEM).setVisible(!(Receiver.headset > 0 || Receiver.docked > 0 ||
Receiver.bluetooth > 0));
		menu.findItem(ANSWER_MENU_ITEM).setVisible(Receiver.call_state ==
UserAgent.UA_STATE_INCOMING_CALL);

		return result;
	}

	public void onClick(DialogInterface dialog, int which)
	{
		if (which == DialogInterface.BUTTON_POSITIVE)
			Receiver.engine(this).transfer(transferText.getText().toString());
	}

	private void transfer() {
		transferText = new InstantAutoCompleteTextView(Receiver.mContext,null);
		transferText.setInputType(InputType.TYPE_CLASS_TEXT |
					InputType.TYPE_TEXT_VARIATION_EMAIL_ADDRESS);

		new AlertDialog.Builder(this)
			.setTitle(Receiver.mContext.getString(R.string.transfer_title))
			.setView(transferText)
			.setPositiveButton(android.R.string.ok, this)
			.setNegativeButton(android.R.string.cancel, this)
			.show();
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		boolean result = super.onOptionsItemSelected(item);
		Intent intent = null;

		switch (item.getItemId()) {
		case HANG_UP_MENU_ITEM:
			Receiver.stopRingtone();
			Receiver.engine(this).rejectcall();
			break;

		case ANSWER_MENU_ITEM:
			Receiver.engine(this).answercall();
			break;

		case HOLD_MENU_ITEM:
			Receiver.engine(this).togglehold();
			break;

		case TRANSFER_MENU_ITEM:
			transfer();
			break;
```

```
                    case MUTE_MENU_ITEM:
                              Receiver.engine(this).togglemute();
                              break;

                    case SPEAKER_MENU_ITEM:
                              Receiver.engine(this).speaker(RtpStreamReceiver.speakermode ==
AudioManager.MODE_NORMAL?
                                                  AudioManager.MODE_IN_CALL:AudioManager.MODE_NORMAL);
                              break;

                    case BLUETOOTH_MENU_ITEM:
                              Receiver.engine(this).togglebluetooth();
                              break;

                    case VIDEO_MENU_ITEM:
                              if (Receiver.call_state == UserAgent.UA_STATE_HOLD) Receiver.engine(this).togglehold();
                              try {
                                        intent = new Intent(this, org.sipdroid.sipua.ui.VideoCamera.class);
                                        startActivity(intent);
                              } catch (ActivityNotFoundException e) {
                              }
                              break;
                    }

                    return result;
          }

          long enabletime;
    KeyguardManager mKeyguardManager;
    KeyguardManager.KeyguardLock mKeyguardLock;
    boolean enabled;

          void disableKeyguard() {
          if (mKeyguardManager == null) {
                  mKeyguardManager = (KeyguardManager) getSystemService(Context.KEYGUARD_SERVICE);
                  mKeyguardLock = mKeyguardManager.newKeyguardLock("Sipdroid");
                  enabled = true;
          }
                    if (enabled) {
                              mKeyguardLock.disableKeyguard();
                              enabled = false;
                              enabletime = SystemClock.elapsedRealtime();
                    }
          }

          void reenableKeyguard() {
                    if (!enabled) {
                                        try {
                                                  if (Integer.parseInt(Build.VERSION.SDK) < 5)
                                                            Thread.sleep(1000);
                                        } catch (InterruptedException e) {
                                        }
                              mKeyguardLock.reenableKeyguard();
                              enabled = true;
                    }
          }

          SipdroidSocket socket;
          RtpSocket rtp_socket;
          Context mContext = this;
          Intent intent;

          @Override
          public void onResume() {
                    super.onResume();
                    if (Integer.parseInt(Build.VERSION.SDK) >= 5 && Integer.parseInt(Build.VERSION.SDK) <= 7)
                              disableKeyguard();
                    if (Receiver.call_state == UserAgent.UA_STATE_INCALL && socket == null &&
Receiver.engine(mContext).getLocalVideo() != 0 && Receiver.engine(mContext).getRemoteVideo() != 0 &&
PreferenceManager.getDefaultSharedPreferences(this).getString(org.sipdroid.sipua.ui.Settings.PREF_SERVER,
org.sipdroid.sipua.ui.Settings.DEFAULT_SERVER).equals(org.sipdroid.sipua.ui.Settings.DEFAULT_SERVER))
                    (new Thread() {
                                        public void run() {
                                                  RtpPacket keepalive = new RtpPacket(new byte[12],0);
                                                  RtpPacket videopacket = new RtpPacket(new byte[1000],0);

                                                  try {
                                                            if (intent == null || rtp_socket == null) {
                                                                      rtp_socket = new RtpSocket(socket = new
SipdroidSocket(Receiver.engine(mContext).getLocalVideo()),

          InetAddress.getByName(Receiver.engine(mContext).getRemoteAddr()),

          Receiver.engine(mContext).getRemoteVideo());
                                                                      sleep(3000);
                                                            } else
                                                                      socket = rtp_socket.getDatagramSocket();
                                                            rtp_socket.getDatagramSocket().setSoTimeout(15000);
                                                  } catch (Exception e) {
                                                            if (!Sipdroid.release) e.printStackTrace();
                                                            return;
                                                  }
                                                  keepalive.setPayloadType(126);
                                                  try {
```

```
                                                            rtp_socket.send(keepalive);
                                                  } catch (Exception e1) {
                                                            return;
                                                  }
                                                  for (;;) {
                                                            try {
                                                                      rtp_socket.receive(videopacket);
                                                            } catch (IOException e) {

            rtp_socket.getDatagramSocket().disconnect();
                                                                      try {
                                                                                rtp_socket.send(keepalive);
                                                                      } catch (IOException e1) {
                                                                                return;
                                                                      }
                                                            }
                                                            if (videopacket.getPayloadLength() > 200) {
                                                                      if (intent != null) {

            intent.putExtra("justplay",true);
                                                                                mHandler.sendEmptyMessage(0);
                                                                      } else {
                                                                                Intent i = new Intent(mContext,
org.sipdroid.sipua.ui.VideoCamera.class);
                                                                                i.putExtra("justplay",true);
                                                                                startActivity(i);
                                                                      }
                                                                      return;
                                                            }
                                                  }
                                        }
                  }).start();
            }

    Handler mHandler = new Handler() {
            public void handleMessage(Message msg) {
                      onResume();
            }
    };

    @Override
            public void onPause() {
                      if (socket != null) {
                                socket.close();
                                socket = null;
                      }
                      super.onPause();
                      if (Integer.parseInt(Build.VERSION.SDK) >= 5 && Integer.parseInt(Build.VERSION.SDK) <= 7)
                                reenableKeyguard();
            }

            @Override
            public void onStart() {
                      super.onStart();
                      if (Integer.parseInt(Build.VERSION.SDK) < 5 || Integer.parseInt(Build.VERSION.SDK) > 7)
                                disableKeyguard();
            }

            @Override
            public void onStop() {
                      super.onStop();
                      if (Integer.parseInt(Build.VERSION.SDK) < 5 || Integer.parseInt(Build.VERSION.SDK) > 7)
                                reenableKeyguard();
            }

}
```

ChangeAccount.java

```
package org.sipdroid.sipua.ui;
import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;

public class ChangeAccount extends Activity {

            public static int getPref(Context context) {
                      return PreferenceManager.getDefaultSharedPreferences(context).getInt(Settings.PREF_ACCOUNT,
Settings.DEFAULT_ACCOUNT);
            }

            @Override
            protected void onCreate(Bundle savedInstanceState) {
                      super.onCreate(savedInstanceState);
                      Editor edit = PreferenceManager.getDefaultSharedPreferences(this).edit();

                      edit.putInt(Settings.PREF_ACCOUNT, Receiver.engine(this).pref = 1-getPref(this));
                      edit.commit();
                      Receiver.engine(this).register();
                      finish();
            }
}
```

CreateAccount.java

```
package org.sipdroid.sipua.ui;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.List;
import java.util.Locale;
import java.util.Random;

import org.sipdroid.sipua.R;
import org.sipdroid.sipua.RegisterAgent;
import org.sipdroid.sipua.SipdroidEngine;

import android.accounts.Account;
import android.accounts.AccountManager;
import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences.Editor;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.preference.PreferenceManager;
import android.text.format.Time;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class CreateAccount extends Dialog {

          Context mContext;

          public CreateAccount(Context context) {
                    super(context);
                    mContext = context;
          }

          static String email,trunkserver,trunkuser,trunkpassword,trunkport;

          public static String isPossible(Context context) {
                    Boolean found = false;
                    email = trunkserver = null;
                    for (int i = 0; i < SipdroidEngine.LINES; i++) {
                              String j = (i!=0?""+i:"");
                              String username =
PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_USERNAME+j, Settings.DEFAULT_USERNAME),
                                        server =
PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_SERVER+j, Settings.DEFAULT_SERVER);
                              if (username.equals("") || server.equals(""))
                                        continue;
                              if (server.contains("pbxes"))
                                        found = true;
                              else if (i == 0 &&

          !PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_PROTOCOL+j,
Settings.DEFAULT_PROTOCOL).equals("tcp") &&

          PreferenceManager.getDefaultSharedPreferences(context).getBoolean(Settings.PREF_3G+j, Settings.DEFAULT_3G) &&
                                                  Receiver.engine(context).isRegistered(i) &&
                                                  Receiver.engine(context).ras[i].CurrentState ==
RegisterAgent.REGISTERED) {
                                        trunkserver = server;
                                        trunkuser = username;
                                        trunkpassword =
PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_PASSWORD+j, Settings.DEFAULT_PASSWORD);
                                        trunkport =
PreferenceManager.getDefaultSharedPreferences(context).getString(Settings.PREF_PORT+j, Settings.DEFAULT_PORT);
                              }
                    }
                    if (found) return null;
        Account[] accounts = AccountManager.get(context).getAccountsByType("com.google");
        for (Account account : accounts) {
             email = account.name;
             break;
        }
        if (email == null) return null;
                    Intent intent = new Intent(Intent.ACTION_SENDTO);
                    intent.setPackage("com.google.android.apps.googlevoice");
                    intent.setData(Uri.fromParts("smsto", "", null));
                    List<ResolveInfo> a =
context.getPackageManager().queryIntentActivities(intent,PackageManager.GET_INTENT_FILTERS);
                    if (a != null && a.size() != 0) {
                              trunkserver = null;
                              return context.getString(R.string.menu_create);
                    }
                    if (trunkserver != null)
                              return "New PBX linked to "+trunkserver;
        return null;
```

```
            }

            String line;

    Handler mHandler = new Handler() {
            public void handleMessage(Message msg) {
                                Toast.makeText(mContext, line, Toast.LENGTH_LONG).show();
                                buttonCancel.setEnabled(true);
                                buttonOK.setEnabled(true);
                                setCancelable(true);
            }
    };

    String generatePassword(int length)
            {
                String availableCharacters = "";
                String password = "";

                // Generate the appropriate character set
                availableCharacters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
                availableCharacters = availableCharacters + "0123456789";

                // Generate the random number generator
                Random selector = new Random();

                // Generate the password
                int i;
                for(i = 0; i < length; i++)
                {
                        password = password + availableCharacters.charAt(selector.nextInt(availableCharacters.length() -
1));
                }

                return password;
            }

            void CreateAccountNow() {
                        buttonCancel.setEnabled(false);
                        buttonOK.setEnabled(false);
                        setCancelable(false);
                        Toast.makeText(mContext, "Please stand by while your account is being created",
Toast.LENGTH_LONG).show();
                (new Thread() {
                                public void run() {
                                        line = "Can't connect to webserver";
                                        try {
                                                String password = generatePassword(8);
                                                String language = Locale.getDefault().toString().substring(0,2);
                                                if (!language.equals("de") && !language.equals("es") &&
!language.equals("fr") &&
                                                                !language.equals("it") &&
!language.equals("ru"))
                                                        if (language.equals("ja"))
                                                                language = "jp";
                                                        else if (language.equals("zh"))
                                                                language = "cn";
                                                        else
                                                                language = "en";
                                                String s =
"https://www1.pbxes.com/config.php?m=register&a=update&f=action&username="+Uri.encode(etName.getText().toString())+"&pas
sword="

            +Uri.encode(etPass.getText().toString())+"&password_confirm="+Uri.encode(etConfirm.getText().toString())+"&la
nguage="+language+"&email="+Uri.encode(email)+"&land="+Uri.encode(Time.getCurrentTimezone())+
                                                "&sipdroid="+Uri.encode(password);
                                                if (trunkserver != null) {
                                                        s =
s+"&trunkserver="+Uri.encode(trunkserver+":"+trunkport)+
                                                                "&trunkuser="+Uri.encode(trunkuser);
                                                }
                                        URL url = new URL(s);
                                        BufferedReader in;
                                                in = new BufferedReader(new InputStreamReader(url.openStream()));
                                                line = in.readLine();
                                                if (line == null) {
                                                        in = new BufferedReader(new
InputStreamReader(url.openStream()));
                                                        line = in.readLine();
                                                }
                                                if (line != null) {
                                                        if (line.equals("OK")) {
                                                                Editor edit =
PreferenceManager.getDefaultSharedPreferences(mContext).edit();
                                                                edit.putString(Settings.PREF_SERVER,
Settings.DEFAULT_SERVER);
                                                                edit.putString(Settings.PREF_USERNAME,
etName.getText()+"-200");
                                                                edit.putString(Settings.PREF_DOMAIN,
Settings.DEFAULT_DOMAIN);
                                                                edit.putString(Settings.PREF_FROMUSER,
Settings.DEFAULT_FROMUSER);
                                                                edit.putString(Settings.PREF_PORT, "5061");
                                                                edit.putString(Settings.PREF_PROTOCOL,
"tcp");
```

```
                                                                        edit.putString(Settings.PREF_PASSWORD,
password);
                                                                        edit.commit();
                                                        Receiver.engine(mContext).updateDNS();
                                                                  Receiver.engine(mContext).halt();
                                                                        Receiver.engine(mContext).StartEngine();
                                                                        dismiss();
                                                                  }
                                                        }
                                              in.close();
                                                 } catch (IOException e) {
                                                        if (!Sipdroid.release) e.printStackTrace();
                                                 }
                                                 mHandler.sendEmptyMessage(0);
                                        }
                              }).start();
                    }

                    EditText etName,etPass,etConfirm;
                    TextView tAdd;
                    Button buttonCancel,buttonOK;

                    @Override
                public void onCreate(Bundle savedInstanceState) {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.create_dialog);
                    setTitle("Create Free Account");
                    buttonOK = (Button) findViewById(R.id.Button01);
                              buttonOK.setOnClickListener(new Button.OnClickListener() {
                                        public void onClick(View v) {
                                                  CreateAccountNow();
                                        }
                              });
                    buttonCancel = (Button) findViewById(R.id.Button02);
                              buttonCancel.setOnClickListener(new Button.OnClickListener() {
                                        public void onClick(View v) {
                                                  dismiss();
                                        }
                              });
                    etName = (EditText) findViewById(R.id.EditText01);
                    etName.setText(email.substring(0,email.indexOf("@")));
                    tAdd = (TextView) findViewById(R.id.Text01);
                    tAdd.setText(email);
                    etPass = (EditText) findViewById(R.id.EditText02);
                    etConfirm = (EditText) findViewById(R.id.EditText03);

                    TextView intro = (TextView) findViewById(R.id.intro);
                    TextView intro2 = (TextView) findViewById(R.id.email);
                    if (trunkserver != null) {
                        intro.setText("To save battery life by utilizing SIP over TCP protocol, a new PBXes account is being offered
to you. It will be automatically linked to your existing "+trunkserver+" account, and therefore get the same password as
your "+trunkserver+" account.");
                        TextView intro3 = (TextView) findViewById(R.id.password);
                        TextView intro4 = (TextView) findViewById(R.id.password_confirm);
                        etPass.setVisibility(View.GONE);
                        etConfirm.setVisibility(View.GONE);
                        etPass.setText(trunkpassword);
                        etConfirm.setText(trunkpassword);
                        intro3.setVisibility(View.GONE);
                        intro4.setVisibility(View.GONE);
                        intro2.setText("Email Address");
                    } else {
                        intro.setText("A new PBXes account will be created. It will be linked to your existing Google Voice account,
and therefore get the same password as your Google Voice account.");
                        intro2.setText("Google Voice Name");
                    }
                }

}
```

InCallScreen.java

```
package org.sipdroid.sipua.ui;
import java.util.HashMap;
import org.sipdroid.media.RtpStreamReceiver;
import org.sipdroid.media.RtpStreamSender;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.UserAgent;
import org.sipdroid.sipua.phone.Call;
import org.sipdroid.sipua.phone.CallCard;
import org.sipdroid.sipua.phone.Phone;
import org.sipdroid.sipua.phone.SlidingCardManager;

import android.content.ContentResolver;
import android.content.Context;
import android.content.pm.ActivityInfo;
import android.content.res.Configuration;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.media.AudioManager;
import android.media.ToneGenerator;
import android.os.Bundle;
import android.os.Handler;
```

```
import android.os.Message;
import android.os.SystemClock;
import android.preference.PreferenceManager;
import android.provider.Settings;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.view.WindowManager;
import android.widget.EditText;
import android.widget.RelativeLayout;
import android.widget.SlidingDrawer;
import android.widget.TextView;

public class InCallScreen extends CallScreen implements View.OnClickListener, SensorEventListener {

           final int MSG_ANSWER = 1;
           final int MSG_ANSWER_SPEAKER = 2;
           final int MSG_BACK = 3;
           final int MSG_TICK = 4;
           final int MSG_POPUP = 5;

           final int SCREEN_OFF_TIMEOUT = 12000;

           CallCard mCallCard;
           Phone ccPhone;
           int oldtimeout;
           SensorManager sensorManager;
    Sensor proximitySensor;
    boolean first;

           void screenOff(boolean off) {
        ContentResolver cr = getContentResolver();

       if (proximitySensor != null)
          return;
       if (off) {
          if (oldtimeout == 0) {
                     oldtimeout = Settings.System.getInt(cr, Settings.System.SCREEN_OFF_TIMEOUT, 60000);
                     Settings.System.putInt(cr, Settings.System.SCREEN_OFF_TIMEOUT, SCREEN_OFF_TIMEOUT);
          }
       } else {
          if (oldtimeout == 0 && Settings.System.getInt(cr, Settings.System.SCREEN_OFF_TIMEOUT, 60000) ==
SCREEN_OFF_TIMEOUT)
                     oldtimeout = 60000;
          if (oldtimeout != 0) {
                     Settings.System.putInt(cr, Settings.System.SCREEN_OFF_TIMEOUT, oldtimeout);
                     oldtimeout = 0;
          }
       }
          }

          @Override
          public void onStop() {
                     super.onStop();
                     mHandler.removeMessages(MSG_BACK);
                     if (Receiver.call_state == UserAgent.UA_STATE_IDLE)
                                finish();
                     sensorManager.unregisterListener(this);
                     started = false;
          }

          @Override
          public void onStart() {
                     super.onStart();
                     if (Receiver.call_state == UserAgent.UA_STATE_IDLE)
                     mHandler.sendEmptyMessageDelayed(MSG_BACK, Receiver.call_end_reason == -1?
                                          2000:5000);
              first = true;
              pactive = false;
              sensorManager.registerListener(this,proximitySensor,SensorManager.SENSOR_DELAY_NORMAL);
              started = true;
          }

          @Override
          public void onPause() {
                     super.onPause();
          if (!Sipdroid.release) Log.i("SipUA:","on pause");
          switch (Receiver.call_state) {
          case UserAgent.UA_STATE_INCOMING_CALL:
                     if (!RtpStreamReceiver.isBluetoothAvailable()) Receiver.moveTop();
                     break;
          case UserAgent.UA_STATE_IDLE:
                     if (Receiver.ccCall != null)
                                mCallCard.displayMainCallStatus(ccPhone,Receiver.ccCall);
                     mHandler.sendEmptyMessageDelayed(MSG_BACK, Receiver.call_end_reason == -1?
                                          2000:5000);
                     break;
          }
                     if (t != null) {
                                running = false;
                                t.interrupt();
                     }
                     screenOff(false);
```

```
                if (mCallCard.mElapsedTime != null) mCallCard.mElapsedTime.stop();
        }

        void moveBack() {
                if (Receiver.ccConn != null && !Receiver.ccConn.isIncoming()) {
                        // after an outgoing call don't fall back to the contact
                        // or call log because it is too easy to dial accidentally from there
                startActivity(Receiver.createHomeIntent());
                }
                onStop();
        }

        Context mContext = this;

        @Override
        public void onResume() {
                super.onResume();
        if (!Sipdroid.release) Log.i("SipUA:","on resume");
                switch (Receiver.call_state) {
                case UserAgent.UA_STATE_INCOMING_CALL:
                        if (Receiver.pstn_state == null || Receiver.pstn_state.equals("IDLE"))
                                if
(PreferenceManager.getDefaultSharedPreferences(mContext).getBoolean(org.sipdroid.sipua.ui.Settings.PREF_AUTO_ON,
org.sipdroid.sipua.ui.Settings.DEFAULT_AUTO_ON) &&
                                                !mKeyguardManager.inKeyguardRestrictedInputMode())
                                        mHandler.sendEmptyMessageDelayed(MSG_ANSWER, 1000);
                                else if
((PreferenceManager.getDefaultSharedPreferences(mContext).getBoolean(org.sipdroid.sipua.ui.Settings.PREF_AUTO_ONDEMAND,
org.sipdroid.sipua.ui.Settings.DEFAULT_AUTO_ONDEMAND) &&

        PreferenceManager.getDefaultSharedPreferences(mContext).getBoolean(org.sipdroid.sipua.ui.Settings.PREF_AUTO_D
EMAND, org.sipdroid.sipua.ui.Settings.DEFAULT_AUTO_DEMAND)) ||

        (PreferenceManager.getDefaultSharedPreferences(mContext).getBoolean(org.sipdroid.sipua.ui.Settings.PREF_AUTO_
HEADSET, org.sipdroid.sipua.ui.Settings.DEFAULT_AUTO_HEADSET) &&
                                                        Receiver.headset > 0))
                                        mHandler.sendEmptyMessageDelayed(MSG_ANSWER_SPEAKER, 10000);
                        break;
                case UserAgent.UA_STATE_INCALL:
                        mDialerDrawer.close();
                        mDialerDrawer.setVisibility(View.VISIBLE);
                        if (Receiver.docked <= 0)
                                screenOff(true);
                        break;
                case UserAgent.UA_STATE_IDLE:
                        if (!mHandler.hasMessages(MSG_BACK))
                                moveBack();
                        break;
                }
                if (Receiver.call_state != UserAgent.UA_STATE_INCALL) {
                        mDialerDrawer.close();
                        mDialerDrawer.setVisibility(View.GONE);
                }
                if (Receiver.ccCall != null) mCallCard.displayMainCallStatus(ccPhone,Receiver.ccCall);
        if (mSlidingCardManager != null) mSlidingCardManager.showPopup();
                mHandler.sendEmptyMessage(MSG_TICK);
                mHandler.sendEmptyMessage(MSG_POPUP);
             if (t == null && Receiver.call_state != UserAgent.UA_STATE_IDLE) {
                        mDigits.setText("");
                        running = true;
                    (t = new Thread() {
                                public void run() {
                                        int len = 0;
                                        long time;
                                        ToneGenerator tg = null;

                                        if (Settings.System.getInt(getContentResolver(),
                                                        Settings.System.DTMF_TONE_WHEN_DIALING, 1)
== 1)
                                                tg = new ToneGenerator(AudioManager.STREAM_VOICE_CALL,
(int)(ToneGenerator.MAX_VOLUME*2*org.sipdroid.sipua.ui.Settings.getEarGain()));
                                        for (;;) {
                                                if (!running) {
                                                        t = null;
                                                        break;
                                                }
                                                if (len != mDigits.getText().length()) {
                                                        time = SystemClock.elapsedRealtime();
                                                        if (tg != null)
tg.startTone(mToneMap.get(mDigits.getText().charAt(len)));

        Receiver.engine(Receiver.mContext).info(mDigits.getText().charAt(len++),250);
                                                        time = 250-(SystemClock.elapsedRealtime()-
time);
                                                        try {
                                                                if (time > 0) sleep(time);
                                                        } catch (InterruptedException e) {
                                                        }
                                                        if (tg != null) tg.stopTone();
                                                        try {
                                                                if (running) sleep(250);
                                                        } catch (InterruptedException e) {
                                                        }
                                                        continue;
                                                }
```

```
                                                                mHandler.sendEmptyMessage(MSG_TICK);
                                                                try {
                                                                          sleep(1000);
                                                                } catch (InterruptedException e) {
                                                                }
                                                      }
                                                      if (tg != null) tg.release();
                                            }
                                  }).start();
                        }
                }

    Handler mHandler = new Handler() {
                public void handleMessage(Message msg) {
                          switch (msg.what) {
                          case MSG_ANSWER:
                          if (Receiver.call_state == UserAgent.UA_STATE_INCOMING_CALL)
                                    answer();
                          break;
                          case MSG_ANSWER_SPEAKER:
                          if (Receiver.call_state == UserAgent.UA_STATE_INCOMING_CALL) {
                                    answer();
                                              Receiver.engine(mContext).speaker(AudioManager.MODE_NORMAL);
                          }
                          break;
                          case MSG_BACK:
                                    moveBack();
                                    break;
                          case MSG_TICK:
                                    mCodec.setText(RtpStreamReceiver.getCodec());
                                    if (RtpStreamReceiver.good != 0) {
                                              if (RtpStreamReceiver.timeout != 0)
                                                        mStats.setText("no data");
                                              else if (RtpStreamSender.m == 2)

                mStats.setText(Math.round(RtpStreamReceiver.loss/RtpStreamReceiver.good*100)+"%loss, "+

                Math.round(RtpStreamReceiver.lost/RtpStreamReceiver.good*100)+"%lost, "+

                Math.round(RtpStreamReceiver.late/RtpStreamReceiver.good*100)+"%late (>"+
                                                                                                (RtpStreamReceiver.jitter-
250*RtpStreamReceiver.mu)/8/RtpStreamReceiver.mu+"ms)");
                                              else

                mStats.setText(Math.round(RtpStreamReceiver.lost/RtpStreamReceiver.good*100)+"%lost, "+

                Math.round(RtpStreamReceiver.late/RtpStreamReceiver.good*100)+"%late (>"+
                                                                                                (RtpStreamReceiver.jitter-
250*RtpStreamReceiver.mu)/8/RtpStreamReceiver.mu+"ms)");
                                              mStats.setVisibility(View.VISIBLE);
                                    } else
                                              mStats.setVisibility(View.GONE);
                                    break;
                          case MSG_POPUP:
                        if (mSlidingCardManager != null) mSlidingCardManager.showPopup();
                                    break;
                          }
                }
    };

                ViewGroup mInCallPanel,mMainFrame;
                SlidingDrawer mDialerDrawer;
                public static SlidingCardManager mSlidingCardManager;
                TextView mStats;
                TextView mCodec;

    public void initInCallScreen() {
        mInCallPanel = (ViewGroup) findViewById(R.id.inCallPanel);
        mMainFrame = (ViewGroup) findViewById(R.id.mainFrame);
        View callCardLayout = getLayoutInflater().inflate(
                    R.layout.call_card_popup,
                    mInCallPanel);
        mCallCard = (CallCard) callCardLayout.findViewById(R.id.callCard);
        mCallCard.reset();

        mSlidingCardManager = new SlidingCardManager();
        mSlidingCardManager.init(ccPhone, this, mMainFrame);
        SlidingCardManager.WindowAttachNotifierView wanv =
            new SlidingCardManager.WindowAttachNotifierView(this);
               wanv.setSlidingCardManager(mSlidingCardManager);
               wanv.setVisibility(View.GONE);
               RelativeLayout.LayoutParams lp = new RelativeLayout.LayoutParams(0, 0);
               mMainFrame.addView(wanv, lp);

               mStats = (TextView) findViewById(R.id.stats);
               mCodec = (TextView) findViewById(R.id.codec);
        mDialerDrawer = (SlidingDrawer) findViewById(R.id.dialer_container);
        mCallCard.displayOnHoldCallStatus(ccPhone,null);
        mCallCard.displayOngoingCallStatus(ccPhone,null);
        if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE)
           mCallCard.updateForLandscapeMode();

        // Have the WindowManager filter out touch events that are "too fat".
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_IGNORE_CHEEK_PRESSES);
```

```
          mDigits = (EditText) findViewById(R.id.digits);
    mDisplayMap.put(R.id.one, '1');
    mDisplayMap.put(R.id.two, '2');
    mDisplayMap.put(R.id.three, '3');
    mDisplayMap.put(R.id.four, '4');
    mDisplayMap.put(R.id.five, '5');
    mDisplayMap.put(R.id.six, '6');
    mDisplayMap.put(R.id.seven, '7');
    mDisplayMap.put(R.id.eight, '8');
    mDisplayMap.put(R.id.nine, '9');
    mDisplayMap.put(R.id.zero, '0');
    mDisplayMap.put(R.id.pound, '#');
    mDisplayMap.put(R.id.star, '*');

    mToneMap.put('1', ToneGenerator.TONE_DTMF_1);
    mToneMap.put('2', ToneGenerator.TONE_DTMF_2);
    mToneMap.put('3', ToneGenerator.TONE_DTMF_3);
    mToneMap.put('4', ToneGenerator.TONE_DTMF_4);
    mToneMap.put('5', ToneGenerator.TONE_DTMF_5);
    mToneMap.put('6', ToneGenerator.TONE_DTMF_6);
    mToneMap.put('7', ToneGenerator.TONE_DTMF_7);
    mToneMap.put('8', ToneGenerator.TONE_DTMF_8);
    mToneMap.put('9', ToneGenerator.TONE_DTMF_9);
    mToneMap.put('0', ToneGenerator.TONE_DTMF_0);
    mToneMap.put('#', ToneGenerator.TONE_DTMF_P);
    mToneMap.put('*', ToneGenerator.TONE_DTMF_S);

    View button;
    for (int viewId : mDisplayMap.keySet()) {
        button = findViewById(viewId);
        button.setOnClickListener(this);
    }
}

       Thread t;
       EditText mDigits;
       boolean running;
       public static boolean started;
private static final HashMap<Integer, Character> mDisplayMap =
    new HashMap<Integer, Character>();
private static final HashMap<Character, Integer> mToneMap =
    new HashMap<Character, Integer>();

       public void onClick(View v) {
    int viewId = v.getId();

    // if the button is recognized
    if (mDisplayMap.containsKey(viewId)) {
                appendDigit(mDisplayMap.get(viewId));
    }
}

void appendDigit(final char c) {
    mDigits.getText().append(c);
}

@Override
       public void onCreate(Bundle icicle) {
                  super.onCreate(icicle);

                  requestWindowFeature(Window.FEATURE_NO_TITLE);
                  setContentView(R.layout.incall);

                  initInCallScreen();

                  sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    proximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);

    if(!android.os.Build.BRAND.equalsIgnoreCase("archos"))
       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_NOSENSOR);
}

       public void reject() {
                  if (Receiver.ccCall != null) {
                            Receiver.stopRingtone();
                            Receiver.ccCall.setState(Call.State.DISCONNECTED);
                            mCallCard.displayMainCallStatus(ccPhone,Receiver.ccCall);
                            mDialerDrawer.close();
                            mDialerDrawer.setVisibility(View.GONE);
              if (mSlidingCardManager != null)
                 mSlidingCardManager.showPopup();
                 }
    (new Thread() {
                            public void run() {
                  Receiver.engine(mContext).rejectcall();
                                 }
                  }).start();
}

       public void answer() {
    (new Thread() {
                            public void run() {
                                       Receiver.engine(mContext).answercall();
                            }
                  }).start();
```

```
                  if (Receiver.ccCall != null) {
                            Receiver.ccCall.setState(Call.State.ACTIVE);
                            Receiver.ccCall.base = SystemClock.elapsedRealtime();
                            mCallCard.displayMainCallStatus(ccPhone,Receiver.ccCall);
                            mDialerDrawer.setVisibility(View.VISIBLE);
                  if (mSlidingCardManager != null)
                     mSlidingCardManager.showPopup();
                     }
               }

               @Override
               public boolean onKeyDown(int keyCode, KeyEvent event) {
            switch (keyCode) {
            case KeyEvent.KEYCODE_MENU:
               if (Receiver.call_state == UserAgent.UA_STATE_INCOMING_CALL && mSlidingCardManager == null) {
                            answer();
                                             return true;
               }
               break;

            case KeyEvent.KEYCODE_CALL:
               switch (Receiver.call_state) {
               case UserAgent.UA_STATE_INCOMING_CALL:
                            answer();
                            break;
               case UserAgent.UA_STATE_INCALL:
               case UserAgent.UA_STATE_HOLD:
                            Receiver.engine(this).togglehold();
                            break;
               }
                // consume KEYCODE_CALL so PhoneWindow doesn't do anything with it
                return true;

            case KeyEvent.KEYCODE_BACK:
               if (mDialerDrawer.isOpened())
                            mDialerDrawer.animateClose();
               else if (Receiver.call_state == UserAgent.UA_STATE_INCOMING_CALL)
                            reject();
                return true;

            case KeyEvent.KEYCODE_CAMERA:
                // Disable the CAMERA button while in-call since it's too
                // easy to press accidentally.
               return true;

            case KeyEvent.KEYCODE_VOLUME_DOWN:
            case KeyEvent.KEYCODE_VOLUME_UP:
               if (Receiver.call_state == UserAgent.UA_STATE_INCOMING_CALL) {
                            Receiver.stopRingtone();
                            return true;
               }
               RtpStreamReceiver.adjust(keyCode,true);
               return true;
            }
            if (Receiver.call_state == UserAgent.UA_STATE_INCALL) {
                     char number = event.getNumber();
                     if (Character.isDigit(number) || number == '*' || number == '#') {
                        appendDigit(number);
                        return true;
                     }
            }
            return super.onKeyDown(keyCode, event);
               }

               @Override
               public boolean onKeyUp(int keyCode, KeyEvent event) {
                            switch (keyCode) {
            case KeyEvent.KEYCODE_VOLUME_DOWN:
            case KeyEvent.KEYCODE_VOLUME_UP:
               RtpStreamReceiver.adjust(keyCode,false);
               return true;
            case KeyEvent.KEYCODE_ENDCALL:
               if (Receiver.pstn_state == null ||
                                             (Receiver.pstn_state.equals("IDLE") && (SystemClock.elapsedRealtime()-
Receiver.pstn_time) > 3000)) {
                                    reject();
                                    return true;
               }
               break;
                            }
                            Receiver.pstn_time = 0;
                            return false;
               }

               public void onAccuracyChanged(Sensor sensor, int accuracy) {
               }

               void setScreenBacklight(float a) {
            WindowManager.LayoutParams lp = getWindow().getAttributes();
            lp.screenBrightness = a;
            getWindow().setAttributes(lp);
               }

               static final float PROXIMITY_THRESHOLD = 5.0f;
               public static boolean pactive;
```

```
          public void onSensorChanged(SensorEvent event) {
                    if (first) {
                              first = false;
                              return;
                    }
                    float distance = event.values[0];
        boolean active = (distance >= 0.0 && distance < PROXIMITY_THRESHOLD && distance <
event.sensor.getMaximumRange());
        pactive = active;
        setScreenBacklight((float) (active?0.1:-1));
          }
}
```

OwnWifi.java

```
package org.sipdroid.sipua.ui;
import org.sipdroid.sipua.UserAgent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

public class OwnWifi extends BroadcastReceiver {

    @Override
          public void onReceive(Context context, Intent intent) {
          if (!Sipdroid.release) Log.i("SipUA:","ownwifi");
          if (Receiver.mContext == null) Receiver.mContext = context;
          if (Receiver.call_state == UserAgent.UA_STATE_IDLE)
                    Receiver.enable_wifi(false);
    }
}
```

Settings.java

```
package org.sipdroid.sipua.ui;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;

import org.sipdroid.codecs.Codecs;
import org.sipdroid.media.RtpStreamReceiver;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.SipdroidEngine;
import org.zoolu.sip.provider.SipStack;

import android.app.AlertDialog;
import android.content.ContentResolver;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.content.DialogInterface.OnClickListener;
import android.content.SharedPreferences.Editor;
import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
import android.os.Bundle;
import android.preference.CheckBoxPreference;
import android.preference.ListPreference;
import android.preference.PreferenceActivity;
import android.preference.PreferenceManager;
import android.text.InputType;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Toast;

public class Settings extends PreferenceActivity implements OnSharedPreferenceChangeListener, OnClickListener {
          // Current settings handler
          private static SharedPreferences settings;
          // Context definition
          private Settings context = null;

          // Path where to store all profiles - !!!should be replaced by some system variable!!!
          private final static String profilePath = "/sdcard/Sipdroid/";
          // Path where is stored the shared preference file - !!!should be replaced by some system variable!!!
          private final String sharedPrefsPath = "/data/data/org.sipdroid.sipua/shared_prefs/";
          // Shared preference file name - !!!should be replaced by some system variable!!!
          private final String sharedPrefsFile = "org.sipdroid.sipua_preferences";
          // List of profile files available on the SD card
          private String[] profileFiles = null;
          // Which profile file to delete
          private int profileToDelete;

          // IDs of the menu items
          private static final int MENU_IMPORT = 0;
          private static final int MENU_DELETE = 1;
          private static final int MENU_EXPORT = 2;

          // All possible values of the PREF_PREF preference (see bellow)
          public static final String VAL_PREF_PSTN = "PSTN";
          public static final String VAL_PREF_SIP = "SIP";
          public static final String VAL_PREF_SIPONLY = "SIPONLY";
          public static final String VAL_PREF_ASK = "ASK";

          /*-
           * ****************************************
```

```
            * **** HOW TO USE SHARED PREFERENCES *****
            * ****************************************
            *
            * If you need to check the existence of the preference key
            *   in this class:              contains(PREF_USERNAME)
            *   in other classes:
           PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).contains(Settings.PREF_USERNAME)
            * If you need to check the existence of the key or check the value of the preference
            *   in this class:              getString(PREF_USERNAME, "").equals("")
            *   in other classes:
           PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).getString(Settings.PREF_USERNAME,
"").equals("")
            * If you need to get the value of the preference
            *   in this class:              getString(PREF_USERNAME, DEFAULT_USERNAME)
            *   in other classes:
           PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).getString(Settings.PREF_USERNAME,
Settings.DEFAULT_USERNAME)
            */

           // Name of the keys in the Preferences XML file
           public static final String PREF_USERNAME = "username";
           public static final String PREF_PASSWORD = "password";
           public static final String PREF_SERVER = "server";
           public static final String PREF_DOMAIN = "domain";
           public static final String PREF_FROMUSER = "fromuser";
           public static final String PREF_PORT = "port";
           public static final String PREF_PROTOCOL = "protocol";
           public static final String PREF_WLAN = "wlan";
           public static final String PREF_3G = "3g";
           public static final String PREF_EDGE = "edge";
           public static final String PREF_VPN = "vpn";
           public static final String PREF_PREF = "pref";
           public static final String PREF_AUTO_ON = "auto_on";
           public static final String PREF_AUTO_ONDEMAND = "auto_on_demand";
           public static final String PREF_AUTO_HEADSET = "auto_headset";
           public static final String PREF_MWI_ENABLED = "MWI_enabled";
           public static final String PREF_REGISTRATION = "registration";
           public static final String PREF_NOTIFY = "notify";
           public static final String PREF_NODATA = "nodata";
           public static final String PREF_SIPRINGTONE = "sipringtone";
           public static final String PREF_SEARCH = "search";
           public static final String PREF_EXCLUDEPAT = "excludepat";
           public static final String PREF_EARGAIN = "eargain";
           public static final String PREF_MICGAIN = "micgain";
           public static final String PREF_HEARGAIN = "heargain";
           public static final String PREF_HMICGAIN = "hmicgain";
           public static final String PREF_OWNWIFI = "ownwifi";
           public static final String PREF_STUN = "stun";
           public static final String PREF_STUN_SERVER = "stun_server";
           public static final String PREF_STUN_SERVER_PORT = "stun_server_port";

           // MMTel configurations (added by mandrajg)
           public static final String PREF_MMTEL = "mmtel";
           public static final String PREF_MMTEL_QVALUE = "mmtel_qvalue";

           // Call recording preferences.
           public static final String PREF_CALLRECORD = "callrecord";

           public static final String PREF_PAR = "par";
           public static final String PREF_IMPROVE = "improve";
           public static final String PREF_POSURL = "posurl";
           public static final String PREF_POS = "pos";
           public static final String PREF_CALLBACK = "callback";
           public static final String PREF_CALLTHRU = "callthru";
           public static final String PREF_CALLTHRU2 = "callthru2";
           public static final String PREF_CODECS = "codecs_new";
           public static final String PREF_DNS = "dns";
           public static final String PREF_VQUALITY = "vquality";
           public static final String PREF_MESSAGE = "vmessage";
           public static final String PREF_BLUETOOTH = "bluetooth";
           public static final String PREF_KEEPON = "keepon";
           public static final String PREF_SELECTWIFI = "selectwifi";
           public static final String PREF_ACCOUNT = "account";

           // Default values of the preferences
           public static final String       DEFAULT_USERNAME = "";
           public static final String       DEFAULT_PASSWORD = "";
           public static final String       DEFAULT_SERVER = "pbxes.org";
           public static final String       DEFAULT_DOMAIN = "";
           public static final String       DEFAULT_FROMUSER = "";
           public static final String       DEFAULT_PORT = "" + SipStack.default_port;
           public static final String       DEFAULT_PROTOCOL = "tcp";
           public static final boolean      DEFAULT_WLAN = true;
           public static final boolean      DEFAULT_3G = false;
           public static final boolean      DEFAULT_EDGE = false;
           public static final boolean      DEFAULT_VPN = false;
           public static final String       DEFAULT_PREF = VAL_PREF_SIP;
           public static final boolean      DEFAULT_AUTO_ON = false;
           public static final boolean      DEFAULT_AUTO_ONDEMAND = false;
           public static final boolean      DEFAULT_AUTO_HEADSET = false;
           public static final boolean      DEFAULT_MWI_ENABLED = true;
           public static final boolean DEFAULT_REGISTRATION = true;
           public static final boolean      DEFAULT_NOTIFY = false;
           public static final boolean      DEFAULT_NODATA = false;
           public static final String       DEFAULT_SIPRINGTONE = "";
```

```
		public static final String	DEFAULT_SEARCH = "";
		public static final String	DEFAULT_EXCLUDEPAT = "";
		public static final float	DEFAULT_EARGAIN = (float) 0.25;
		public static final float	DEFAULT_MICGAIN = (float) 0.25;
		public static final float	DEFAULT_HEARGAIN = (float) 0.25;
		public static final float	DEFAULT_HMICGAIN = (float) 1.0;
		public static final boolean	DEFAULT_OWNWIFI = false;
		public static final boolean	DEFAULT_STUN = false;
		public static final String	DEFAULT_STUN_SERVER = "stun.ekiga.net";
		public static final String	DEFAULT_STUN_SERVER_PORT = "3478";

		// MMTel configuration (added by mandrajg)
		public static final boolean	DEFAULT_MMTEL = false;
		public static final String	DEFAULT_MMTEL_QVALUE = "1.00";

		// Call recording preferences.
		public static final boolean DEFAULT_CALLRECORD = false;

		public static final boolean	DEFAULT_PAR = false;
		public static final boolean	DEFAULT_IMPROVE = false;
		public static final String	DEFAULT_POSURL = "";
		public static final boolean	DEFAULT_POS = false;
		public static final boolean	DEFAULT_CALLBACK = false;
		public static final boolean	DEFAULT_CALLTHRU = false;
		public static final String	DEFAULT_CALLTHRU2 = "";
		public static final String	DEFAULT_CODECS = null;
		public static final String	DEFAULT_DNS = "";
		public static final String  DEFAULT_VQUALITY = "low";
		public static final boolean DEFAULT_MESSAGE = false;
		public static final boolean DEFAULT_BLUETOOTH = false;
		public static final boolean DEFAULT_KEEPON = false;
		public static final boolean DEFAULT_SELECTWIFI = false;
		public static final int     DEFAULT_ACCOUNT = 0;

		// An other preference keys (not in the Preferences XML file)
		public static final String PREF_OLDVALID = "oldvalid";
		public static final String PREF_SETMODE = "setmode";
		public static final String PREF_OLDVIBRATE = "oldvibrate";
		public static final String PREF_OLDVIBRATE2 = "oldvibrate2";
		public static final String PREF_OLDPOLICY = "oldpolicy";
		public static final String PREF_OLDRING = "oldring";
		public static final String PREF_AUTO_DEMAND = "auto_demand";
		public static final String PREF_WIFI_DISABLED = "wifi_disabled";
		public static final String PREF_ON_VPN = "on_vpn";
		public static final String PREF_NODEFAULT = "nodefault";
		public static final String PREF_NOPORT = "noport";
		public static final String PREF_ON = "on";
		public static final String PREF_PREFIX = "prefix";
		public static final String PREF_COMPRESSION = "compression";
		//public static final String PREF_RINGMODEx = "ringmodeX";
		//public static final String PREF_VOLUMEx = "volumeX";

		// Default values of the other preferences
		public static final boolean	DEFAULT_OLDVALID = false;
		public static final boolean	DEFAULT_SETMODE = false;
		public static final int			DEFAULT_OLDVIBRATE = 0;
		public static final int			DEFAULT_OLDVIBRATE2 = 0;
		public static final int			DEFAULT_OLDPOLICY = 0;
		public static final int			DEFAULT_OLDRING = 0;
		public static final boolean	DEFAULT_AUTO_DEMAND = false;
		public static final boolean	DEFAULT_WIFI_DISABLED = false;
		public static final boolean DEFAULT_ON_VPN = false;
		public static final boolean	DEFAULT_NODEFAULT = false;
		public static final boolean DEFAULT_NOPORT = false;
		public static final boolean	DEFAULT_ON = false;
		public static final String	DEFAULT_PREFIX = "";
		public static final String	DEFAULT_COMPRESSION = null;
		//public static final String	DEFAULT_RINGTONEx = "";
		//public static final String	DEFAULT_VOLUMEx = "";

		public static float getEarGain() {
			try {
				return
Float.valueOf(PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).getString(Receiver.headset > 0 ?
PREF_HEARGAIN : PREF_EARGAIN, "" + DEFAULT_EARGAIN));
			} catch (NumberFormatException i) {
				return DEFAULT_EARGAIN;
			}
		}

		public static float getMicGain() {
			if (Receiver.headset > 0 || Receiver.bluetooth > 0) {
				try {
					return
Float.valueOf(PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).getString(PREF_HMICGAIN, "" +
DEFAULT_HMICGAIN));
				} catch (NumberFormatException i) {
					return DEFAULT_HMICGAIN;
				}
			}

			try {
				return
Float.valueOf(PreferenceManager.getDefaultSharedPreferences(Receiver.mContext).getString(PREF_MICGAIN, "" +
DEFAULT_MICGAIN));
```

```
                } catch (NumberFormatException i) {
                        return DEFAULT_MICGAIN;
                }
        }

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);

        if (Receiver.mContext == null) Receiver.mContext = this;
                addPreferencesFromResource(R.xml.preferences);
//              setDefaultValues();
                Codecs.check();
        }

        void reload() {
                setPreferenceScreen(null);
                addPreferencesFromResource(R.xml.preferences);
        }

        private void setDefaultValues() {
                settings = getSharedPreferences(sharedPrefsFile, MODE_PRIVATE);

                for (int i = 0; i < SipdroidEngine.LINES; i++) {
                        String j = (i!=0?""+i:"");
                        if (settings.getString(PREF_SERVER+j, "").equals("")) {
                                CheckBoxPreference cb = (CheckBoxPreference)
getPreferenceScreen().findPreference(PREF_WLAN+j);
                                cb.setChecked(true);
                                Editor edit = settings.edit();

                                edit.putString(PREF_PORT+j, "5061");
                                edit.putString(PREF_SERVER+j, DEFAULT_SERVER);
                                edit.putString(PREF_PREF+j, DEFAULT_PREF);
                                edit.putString(PREF_PROTOCOL+j, DEFAULT_PROTOCOL);
                                edit.commit();
                Receiver.engine(this).updateDNS();
                reload();
                        }
                }
                if (settings.getString(PREF_STUN_SERVER, "").equals("")) {
                        Editor edit = settings.edit();

                        edit.putString(PREF_STUN_SERVER, DEFAULT_STUN_SERVER);
                        edit.putString(PREF_STUN_SERVER_PORT, DEFAULT_STUN_SERVER_PORT);

                        edit.commit();
                        reload();
                }

                if (! settings.contains(PREF_MWI_ENABLED)) {
                        CheckBoxPreference cb = (CheckBoxPreference)
getPreferenceScreen().findPreference(PREF_MWI_ENABLED);
                        cb.setChecked(true);
                }
                if (! settings.contains(PREF_REGISTRATION)) {
                        CheckBoxPreference cb = (CheckBoxPreference)
getPreferenceScreen().findPreference(PREF_REGISTRATION);
                        cb.setChecked(true);
                }
                if (Sipdroid.market) {
                        CheckBoxPreference cb = (CheckBoxPreference)
getPreferenceScreen().findPreference(PREF_3G);
                        cb.setChecked(false);
                        CheckBoxPreference cb2 = (CheckBoxPreference)
getPreferenceScreen().findPreference(PREF_EDGE);
                        cb2.setChecked(false);
                        getPreferenceScreen().findPreference(PREF_3G).setEnabled(false);
                        getPreferenceScreen().findPreference(PREF_EDGE).setEnabled(false);
                }

                settings.registerOnSharedPreferenceChangeListener(this);

                updateSummaries();
        }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
            menu.add(0, MENU_IMPORT, 0,
getString(R.string.settings_profile_menu_import)).setIcon(android.R.drawable.ic_menu_upload);
            menu.add(0, MENU_EXPORT, 0,
getString(R.string.settings_profile_menu_export)).setIcon(android.R.drawable.ic_menu_save);
            menu.add(0, MENU_DELETE, 0,
getString(R.string.settings_profile_menu_delete)).setIcon(android.R.drawable.ic_menu_delete);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        context = this;

        switch (item.getItemId()) {
         case MENU_IMPORT:
                // Get the content of the directory
                profileFiles = getProfileList();
                if (profileFiles != null && profileFiles.length > 0) {
```

```
                                // Show dialog with the files
                                          new AlertDialog.Builder(this)
                                          .setTitle(getString(R.string.settings_profile_dialog_profiles_title))
                                          .setIcon(android.R.drawable.ic_menu_upload)
                                          .setItems(profileFiles, profileOnClick)
                                          .show();
                          } else {
                               Toast.makeText(this, "No profile found.", Toast.LENGTH_SHORT).show();
                          }
                      return true;

                  case MENU_EXPORT:
                          exportSettings();
                          break;

                  case MENU_DELETE:
                          // Get the content of the directory
                          profileFiles = getProfileList();
                          new AlertDialog.Builder(this)
                      .setTitle(getString(R.string.settings_profile_dialog_delete_title))
                      .setIcon(android.R.drawable.ic_menu_delete)
                                  .setItems(profileFiles, new DialogInterface.OnClickListener() {
                                          // Ask the user to be sure to delete it
                                          public void onClick(DialogInterface dialog, int whichItem) {
                                          profileToDelete = whichItem;
                                                   new AlertDialog.Builder(context)
                              .setIcon(android.R.drawable.ic_dialog_alert)
                              .setTitle(getString(R.string.settings_profile_dialog_delete_title))
                              .setMessage(getString(R.string.settings_profile_dialog_delete_text, profileFiles[whichItem]))
                              .setPositiveButton(android.R.string.ok, deleteOkButtonClick)
                              .setNegativeButton(android.R.string.cancel, null)
                              .show();
                                          }
                                  })
                      .show();
                      return true;
              }

              return false;
          }

          public static String[] getProfileList() {
                 File dir = new File(profilePath);
                 return dir.list();
          }

          private String getProfileNameString() {
                 return getProfileNameString(settings);
          }

          public static String getProfileNameString(SharedPreferences s) {
                 String provider = s.getString(PREF_SERVER, DEFAULT_SERVER);

                 if (! s.getString(PREF_DOMAIN, "").equals("")) {
                          provider = s.getString(PREF_DOMAIN, DEFAULT_DOMAIN);
                 }

                 return s.getString(PREF_USERNAME, DEFAULT_USERNAME) + "@" + provider;
          }

          private void exportSettings() {
                          if (! settings.getString(PREF_USERNAME, "").equals("") && ! settings.getString(PREF_SERVER,
"").equals(""))
                            try {
                               // Create the directory for the profiles
                               new File(profilePath).mkdirs();

                               // Copy shared preference file on the SD card
                               copyFile(new File(sharedPrefsPath + sharedPrefsFile + ".xml"), new File(profilePath +
getProfileNameString()));
                            } catch (Exception e) {
                                Toast.makeText(this, getString(R.string.settings_profile_export_error),
Toast.LENGTH_SHORT).show();
                            }
          }

                 private OnClickListener profileOnClick = new DialogInterface.OnClickListener() {
                          public void onClick(DialogInterface dialog, int whichItem) {
                                          int set = updateSleepPolicy();
                                          boolean message = settings.getBoolean(PREF_MESSAGE, DEFAULT_MESSAGE);

                                          try {
                                                   copyFile(new File(profilePath + profileFiles[whichItem]), new
File(sharedPrefsPath + sharedPrefsFile + ".xml"));
                     } catch (Exception e) {
                              Toast.makeText(context, getString(R.string.settings_profile_import_error), Toast.LENGTH_SHORT).show();
                              return;
                     }

                                          settings.unregisterOnSharedPreferenceChangeListener(context);
                                          setDefaultValues();

                                  // Restart the engine
                                  Receiver.engine(context).halt();
                                          Receiver.engine(context).StartEngine();
```

```
                                reload();
                                settings.registerOnSharedPreferenceChangeListener(context);
                                updateSummaries();
                                if (set != updateSleepPolicy())
                                        updateSleep();
                                if (message) {
                                Editor edit = settings.edit();
                                edit.putBoolean(PREF_MESSAGE, true);
                                edit.commit();
                                }
                        }
                };

                private OnClickListener deleteOkButtonClick = new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {
                File profile = new File(profilePath + profileFiles[profileToDelete]);
                boolean rv = false;
                // Check if the file exists and try to delete it
                if (profile.exists()) {
                        rv = profile.delete();
                }
                if (rv) {
                        Toast.makeText(context, getString(R.string.settings_profile_delete_confirmation),
Toast.LENGTH_SHORT).show();
                } else {
                        Toast.makeText(context, getString(R.string.settings_profile_delete_error),
Toast.LENGTH_SHORT).show();
                }
                        }
                };

    public void copyFile(File in, File out) throws Exception {
        FileInputStream  fis = new FileInputStream(in);
        FileOutputStream fos = new FileOutputStream(out);
        try {
            byte[] buf = new byte[1024];
            int i = 0;
            while ((i = fis.read(buf)) != -1) {
                fos.write(buf, 0, i);
            }
        } catch (Exception e) {
            throw e;
        } finally {
            if (fis != null) fis.close();
            if (fos != null) fos.close();
        }
    }

                @Override
                public void onDestroy()          {
                        super.onDestroy();

//                      settings.unregisterOnSharedPreferenceChangeListener(this);
                }

                EditText transferText;
                String mKey;

    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
                if (!Thread.currentThread().getName().equals("main"))
                        return;
                        if (key.startsWith(PREF_PORT) && sharedPreferences.getString(key, DEFAULT_PORT).equals("0")) {
                                Editor edit = sharedPreferences.edit();
                        edit.putString(key, DEFAULT_PORT);
                        edit.commit();

                        transferText = new InstantAutoCompleteTextView(this,null);
                                transferText.setInputType(InputType.TYPE_CLASS_NUMBER);
                                mKey = key;

                                new AlertDialog.Builder(this)
                                .setTitle(Receiver.mContext.getString(R.string.settings_port))
                                .setView(transferText)
                                .setPositiveButton(android.R.string.ok, this)
                                .show();
                                return;
                        } else if (key.startsWith(PREF_SERVER)) {
                        Editor edit = sharedPreferences.edit();
                        for (int i = 0; i < SipdroidEngine.LINES; i++) {
                                edit.putString(PREF_DNS+i, DEFAULT_DNS);
                                String j = (i!=0?""+i:"");
                                if (key.equals(PREF_SERVER+j)) {
                                        ListPreference lp = (ListPreference)
getPreferenceScreen().findPreference(PREF_PROTOCOL+j);
                                        lp.setValue(sharedPreferences.getString(PREF_SERVER+j,
DEFAULT_SERVER).equals(DEFAULT_SERVER) ? "tcp" : "udp");
                                        lp = (ListPreference) getPreferenceScreen().findPreference(PREF_PORT+j);
                                        lp.setValue(sharedPreferences.getString(PREF_SERVER+j,
DEFAULT_SERVER).equals(DEFAULT_SERVER) ? "5061" : DEFAULT_PORT);
                                }
                        }
                        edit.commit();
                Receiver.engine(this).updateDNS();
                Checkin.checkin(false);
```

```
        } else if (sharedPreferences.getBoolean(PREF_CALLBACK, DEFAULT_CALLBACK) &&
sharedPreferences.getBoolean(PREF_CALLTHRU, DEFAULT_CALLTHRU)) {
                    CheckBoxPreference cb = (CheckBoxPreference)
getPreferenceScreen().findPreference(key.equals(PREF_CALLBACK) ? PREF_CALLTHRU : PREF_CALLBACK);
                    cb.setChecked(false);
               } else if (key.startsWith(PREF_WLAN) ||
                                key.startsWith(PREF_3G) ||
                                key.startsWith(PREF_EDGE) ||
                                key.startsWith(PREF_USERNAME) ||
                                key.startsWith(PREF_PASSWORD) ||
                                key.startsWith(PREF_DOMAIN) ||
                                key.startsWith(PREF_SERVER) ||
                                key.startsWith(PREF_PORT) ||
                                key.equals(PREF_STUN) ||
                                key.equals(PREF_STUN_SERVER) ||
                                key.equals(PREF_STUN_SERVER_PORT) ||
                                key.equals(PREF_MMTEL) ||                                   // (added by mandrajg)
                                key.equals(PREF_MMTEL_QVALUE) || // (added by mandrajg)
                                key.startsWith(PREF_PROTOCOL) ||
                                key.startsWith(PREF_VPN) ||
                                key.equals(PREF_POS) ||
                                key.equals(PREF_POSURL) ||
                                key.startsWith(PREF_FROMUSER) ||
                                key.equals(PREF_AUTO_ONDEMAND) ||
                                key.equals(PREF_MWI_ENABLED) ||
                                key.equals(PREF_REGISTRATION) ||
                                key.equals(PREF_KEEPON)) {
          Receiver.engine(this).halt();
                    Receiver.engine(this).StartEngine();
                    }
                    if (key.startsWith(PREF_WLAN) || key.startsWith(PREF_3G) || key.startsWith(PREF_EDGE) ||
key.startsWith(PREF_OWNWIFI)) {
                                updateSleep();
                    }

                    updateSummaries();
    }

    int updateSleepPolicy() {
        ContentResolver cr = getContentResolver();
                    int get = android.provider.Settings.System.getInt(cr,
android.provider.Settings.System.WIFI_SLEEP_POLICY, -1);
                    int set = get;
                    boolean wlan = false,g3 = true,valid = false;
                    for (int i = 0; i < SipdroidEngine.LINES; i++) {
                              String j = (i!=0?""+i:"");
                              if (!settings.getString(PREF_USERNAME+j, "").equals("") &&
                                                  !settings.getString(PREF_SERVER+j, "").equals("")) {
                                        valid = true;
                                        wlan |= settings.getBoolean(PREF_WLAN+j, DEFAULT_WLAN);
                                        g3 &= settings.getBoolean(PREF_3G+j, DEFAULT_3G) ||
                                                  settings.getBoolean(PREF_EDGE+j, DEFAULT_EDGE);
                              }
                    }
                    boolean ownwifi = settings.getBoolean(PREF_OWNWIFI, DEFAULT_OWNWIFI);

                    if (g3 && valid && !ownwifi) {
                              set = android.provider.Settings.System.WIFI_SLEEP_POLICY_DEFAULT;
                    } else if (wlan || ownwifi) {
                              set = android.provider.Settings.System.WIFI_SLEEP_POLICY_NEVER;
                    }
                    return set;
    }

          void updateSleep() {
        ContentResolver cr = getContentResolver();
                    int get = android.provider.Settings.System.getInt(cr,
android.provider.Settings.System.WIFI_SLEEP_POLICY, -1);
                    int set = updateSleepPolicy();

                    if (set != get) {
                              Toast.makeText(this, set == android.provider.Settings.System.WIFI_SLEEP_POLICY_DEFAULT?
                                                  R.string.settings_policy_default:R.string.settings_policy_never,
Toast.LENGTH_LONG).show();
                              android.provider.Settings.System.putInt(cr,
android.provider.Settings.System.WIFI_SLEEP_POLICY, set);
                    }
          }

          void fill(String pref,String def,int val,int disp) {
//        for (int i = 0; i < getResources().getStringArray(val).length; i++) {
//        if (settings.getString(pref, def).equals(getResources().getStringArray(val)[i])) {
//                  getPreferenceScreen().findPreference(pref).setSummary(getResources().getStringArray(disp)[i]);
//        }
//        }
    }

          public void updateSummaries() {
//        getPreferenceScreen().findPreference(PREF_STUN_SERVER).setSummary(settings.getString(PREF_STUN_SERVER,
DEFAULT_STUN_SERVER));
//
          getPreferenceScreen().findPreference(PREF_STUN_SERVER_PORT).setSummary(settings.getString(PREF_STUN_SERVER_PO
RT, DEFAULT_STUN_SERVER_PORT));

          // MMTel settings (added by mandrajg)
```

```
//        getPreferenceScreen().findPreference(PREF_MMTEL_QVALUE).setSummary(settings.getString(PREF_MMTEL_QVALUE,
DEFAULT_MMTEL_QVALUE));

          getPreferenceScreen().findPreference(PREF_DOMAIN+j).setSummary(settings.getString(PREF_DOMAIN+j,
DEFAULT_DOMAIN));
DEFAULT_FROMUSER));
//                  }
//                  getPreferenceScreen().findPreference(PREF_PORT+j).setSummary(settings.getString(PREF_PORT+j,
DEFAULT_PORT));
//
          getPreferenceScreen().findPreference(PREF_PROTOCOL+j).setSummary(settings.getString(PREF_PROTOCOL+j,
//                            settings.getString(PREF_SERVER+j, DEFAULT_SERVER).equals(DEFAULT_SERVER) ? "tcp" :
"udp").toUpperCase());
//
          getPreferenceScreen().findPreference(PREF_ACCOUNT+j).setSummary(username.equals("")||server.equals("")?getRes
ources().getString(R.string.settings_line)+" "+(i+1):username+"@"+server);
//        }

//        getPreferenceScreen().findPreference(PREF_SEARCH).setSummary(settings.getString(PREF_SEARCH,
DEFAULT_SEARCH));
//        getPreferenceScreen().findPreference(PREF_EXCLUDEPAT).setSummary(settings.getString(PREF_EXCLUDEPAT,
DEFAULT_EXCLUDEPAT));
//        getPreferenceScreen().findPreference(PREF_POSURL).setSummary(settings.getString(PREF_POSURL,
DEFAULT_POSURL));
//        getPreferenceScreen().findPreference(PREF_CALLTHRU2).setSummary(settings.getString(PREF_CALLTHRU2,
DEFAULT_CALLTHRU2));
 (false);
//                  getPreferenceScreen().findPreference(PREF_STUN_SERVER_PORT).setEnabled(false);
//        }


          public void onClick(DialogInterface arg0, int arg1) {
                    Editor edit = settings.edit();
                    edit.putString(mKey, transferText.getText().toString());
                    edit.commit();
          }
```

RegisterService.java

```
package org.sipdroid.sipua.ui;
import org.sipdroid.media.RtpStreamReceiver;
import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.ConnectivityManager;
import android.net.wifi.WifiManager;
import android.os.IBinder;

public class RegisterService extends Service {
          Receiver m_receiver;
          Caller m_caller;

    public void onDestroy() {
                    super.onDestroy();
                    if (m_receiver != null) {
                              unregisterReceiver(m_receiver);
                              m_receiver = null;
                    }
                    Receiver.alarm(0, OneShotAlarm2.class);
          }

    @Override
    public void onCreate() {
          super.onCreate();
          if (Receiver.mContext == null) Receiver.mContext = this;
        if (m_receiver == null) {
                              IntentFilter intentfilter = new IntentFilter();
                              intentfilter.addAction(ConnectivityManager.CONNECTIVITY_ACTION);
                              intentfilter.addAction(Receiver.ACTION_DATA_STATE_CHANGED);
                              intentfilter.addAction(Receiver.ACTION_PHONE_STATE_CHANGED);
                              intentfilter.addAction(Receiver.ACTION_DOCK_EVENT);
                              intentfilter.addAction(Intent.ACTION_HEADSET_PLUG);
                              intentfilter.addAction(Intent.ACTION_USER_PRESENT);
                              intentfilter.addAction(Intent.ACTION_SCREEN_OFF);
                              intentfilter.addAction(Intent.ACTION_SCREEN_ON);
                              intentfilter.addAction(Receiver.ACTION_VPN_CONNECTIVITY);
                              intentfilter.addAction(Receiver.ACTION_SCO_AUDIO_STATE_CHANGED);
                              intentfilter.addAction(WifiManager.WIFI_STATE_CHANGED_ACTION);
                              intentfilter.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
                  registerReceiver(m_receiver = new Receiver(), intentfilter);
                  intentfilter = new IntentFilter();
        }
        Receiver.engine(this).isRegistered();
        RtpStreamReceiver.restoreSettings();
    }

    @Override
    public void onStart(Intent intent, int id) {
         super.onStart(intent,id);
         Receiver.alarm(10*60, OneShotAlarm2.class);
    }

          @Override
          public IBinder onBind(Intent arg0) {
                    return null;
```

```
	}

}
```

SIPUri.java

```
package org.sipdroid.sipua.ui;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.SipdroidEngine;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnCancelListener;
import android.net.Uri;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Window;

public class SIPUri extends Activity {

	void call(String target) {
		if (!Receiver.engine(this).call(target,true)) {
			new AlertDialog.Builder(this)
			.setMessage(R.string.notfast)
			.setTitle(R.string.app_name)
			.setIcon(R.drawable.icon22)
			.setCancelable(true)
			.setOnCancelListener(new OnCancelListener() {
				public void onCancel(DialogInterface dialog) {
					finish();
				}
			})
			.show();
		} else
			finish();
	}

	/* (non-Javadoc)
	 * @see android.app.Activity#onCreate(android.os.Bundle)
	 */
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
	if (Receiver.mContext == null) Receiver.mContext = this;

		requestWindowFeature(Window.FEATURE_NO_TITLE);

		Sipdroid.on(this,true);
		Uri uri = getIntent().getData();
		String target;
		if (uri.getScheme().equals("sip") || uri.getScheme().equals("sipdroid"))
			target = uri.getSchemeSpecificPart();
		else {
			if (uri.getAuthority().equals("aim") ||
					uri.getAuthority().equals("yahoo") ||
					uri.getAuthority().equals("icq") ||
					uri.getAuthority().equals("gtalk") ||
					uri.getAuthority().equals("msn"))
				target = uri.getLastPathSegment().replaceAll("@","_at_") + "@" +
uri.getAuthority() + ".gtalk2voip.com";
			else if (uri.getAuthority().equals("skype"))
				target = uri.getLastPathSegment() + "@" + uri.getAuthority();
			else
				target = uri.getLastPathSegment();
		}
		if (!Sipdroid.release) Log.v("SIPUri", "sip uri: " + target);
		if (!target.contains("@") &&
PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_PREF,
Settings.DEFAULT_PREF).equals(Settings.VAL_PREF_ASK)) {
			final String t = target;
			String items[] = {getString(R.string.pstn_name)};
			for (int p = 0; p < SipdroidEngine.LINES; p++)
				if (Receiver.isFast(p) ||
(PreferenceManager.getDefaultSharedPreferences(this).getBoolean(Settings.PREF_CALLBACK, Settings.DEFAULT_CALLBACK) &&

	PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_POSURL,
Settings.DEFAULT_POSURL).length() > 0)) {
					items = new String[2];
					items[0] = getString(R.string.app_name);
					items[1] = getString(R.string.pstn_name);
					break;
				}
			final String fitems[] = items;
			new AlertDialog.Builder(this)
			.setIcon(R.drawable.icon22)
			.setTitle(target)
	  .setItems(items, new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int whichButton) {
			if (fitems[whichButton].equals(getString(R.string.app_name)))
				call(t);
			else {
				PSTN.callPSTN("sip:"+t);
				finish();
			}
```

```
                    }
                })
                                .setOnCancelListener(new OnCancelListener() {
                                        public void onCancel(DialogInterface dialog) {
                                                finish();
                                        }
                                })
                                .show();
                        } else
                                call(target);
        }

            @Override
            public void onPause() {
                super.onPause();
                finish();
            }

}
```

SIP.java

```
package org.sipdroid.sipua.ui;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.SystemClock;
import android.preference.PreferenceManager;

public class SIP extends Activity {

        void callPSTN(String uri) {
                String number;

                if (uri.indexOf(":") >= 0) {
                        number = uri.substring(uri.indexOf(":")+1);
                        if (!number.equals("")) {
                        Intent intent = new Intent(Intent.ACTION_CALL,
                                Uri.fromParts(Uri.decode(number).contains("@")?"sipdroid":"tel",
Uri.decode(number)+

        (PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_PREF,
Settings.DEFAULT_PREF).equals(Settings.VAL_PREF_PSTN) ? "+" : ""), null));
                        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        Caller.noexclude = SystemClock.elapsedRealtime();
                        startActivity(intent);
                           }
                }
        }

        @Override
        public void onCreate(Bundle saved) {
                super.onCreate(saved);
                Intent intent;
                Uri uri;
                Sipdroid.on(this,true);
                if ((intent = getIntent()) != null
                        && (uri = intent.getData()) != null)
                                callPSTN(uri.toString());
                finish();
        }
}
```

Sipdroid.java

```
package org.sipdroid.sipua.ui;
import java.util.ArrayList;
import java.util.List;
import org.sipdroid.sipua.R;
import org.sipdroid.sipua.SipdroidEngine;
import org.sipdroid.sipua.UserAgent;
import org.zoolu.tools.Random;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.ContentResolver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.DialogInterface.OnDismissListener;
import android.content.SharedPreferences.Editor;
import android.content.pm.PackageManager.NameNotFoundException;
import android.database.Cursor;
import android.database.CursorWrapper;
import android.os.Build;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.provider.CallLog.Calls;
import android.provider.Contacts.People;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
```

```
import android.view.ViewGroup;
import android.view.Window;
import android.view.View.OnKeyListener;
import android.widget.AdapterView;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.CursorAdapter;
import android.widget.Filterable;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

/////////////////////////////////////////////////////////////////////
// this the main activity of Sipdroid
// for modifying it additional terms according to section 7, GPL apply
// see ADDITIONAL_TERMS.txt
/////////////////////////////////////////////////////////////////////
public class Sipdroid extends Activity implements OnDismissListener {

	public static final boolean release = true;
	public static final boolean market = false;

	/* Following the menu item constants which will be used for menu creation */
	public static final int FIRST_MENU_ID = Menu.FIRST;
	public static final int CONFIGURE_MENU_ITEM = FIRST_MENU_ID + 1;
	public static final int ABOUT_MENU_ITEM = FIRST_MENU_ID + 2;
	public static final int EXIT_MENU_ITEM = FIRST_MENU_ID + 3;

	private static AlertDialog m_AlertDlg;
	AutoCompleteTextView sip_uri_box2;
	Button createButton;

	@Override
	public void onStart() {
		super.onStart();
		Receiver.engine(this).registerMore();
	    ContentResolver content = getContentResolver();
	    Cursor cursor = content.query(Calls.CONTENT_URI,
	            PROJECTION, Calls.NUMBER+" like ?", new String[] { "%@%" }, Calls.DEFAULT_SORT_ORDER);
	    CallsAdapter adapter = new CallsAdapter(this, cursor);
	    sip_uri_box2.setAdapter(adapter);
	}

	public static class CallsCursor extends CursorWrapper {
		List<String> list;

		public int getCount() {
			return list.size();
		}

		public String getString(int i) {
			return list.get(getPosition());
		}

		public CallsCursor(Cursor cursor) {
			super(cursor);
			list = new ArrayList<String>();
			for (int i = 0; i < cursor.getCount(); i++) {
				moveToPosition(i);
		        String phoneNumber = super.getString(1);
		        String cachedName = super.getString(2);
		        if (cachedName != null && cachedName.trim().length() > 0)
		           phoneNumber += " <" + cachedName + ">";
		        if (list.contains(phoneNumber)) continue;
				list.add(phoneNumber);
			   }
			   moveToFirst();
		}

	}

	public static class CallsAdapter extends CursorAdapter implements Filterable {
	    public CallsAdapter(Context context, Cursor c) {
	        super(context, c);
	        mContent = context.getContentResolver();
	    }

	    public View newView(Context context, Cursor cursor, ViewGroup parent) {
	        final LayoutInflater inflater = LayoutInflater.from(context);
	        final TextView view = (TextView) inflater.inflate(
	                android.R.layout.simple_dropdown_item_1line, parent, false);
	           String phoneNumber = cursor.getString(1);
	        view.setText(phoneNumber);
	        return view;
	    }

	    @Override
	    public void bindView(View view, Context context, Cursor cursor) {
	           String phoneNumber = cursor.getString(1);
	        ((TextView) view).setText(phoneNumber);
	    }

	    @Override
	    public String convertToString(Cursor cursor) {
	           String phoneNumber = cursor.getString(1);
	           if (phoneNumber.contains(" <"))
```

```
                        phoneNumber = phoneNumber.substring(0,phoneNumber.indexOf(" <"));
                return phoneNumber;
            }

            @Override
            public Cursor runQueryOnBackgroundThread(CharSequence constraint) {
                if (getFilterQueryProvider() != null) {
                    return new CallsCursor(getFilterQueryProvider().runQuery(constraint));
                }

                StringBuilder buffer;
                String[] args;
                buffer = new StringBuilder();
                buffer.append(Calls.NUMBER);
                buffer.append(" LIKE ? OR ");
                buffer.append(Calls.CACHED_NAME);
                buffer.append(" LIKE ?");
                String arg = "%" + (constraint != null && constraint.length() > 0?
                                constraint.toString() : "@") + "%";
                args = new String[] { arg, arg};

                return new CallsCursor(mContent.query(Calls.CONTENT_URI, PROJECTION,
                        buffer.toString(), args,
                        Calls.NUMBER + " asc"));
            }

            private ContentResolver mContent;
        }

        private static final String[] PROJECTION = new String[] {
    Calls._ID,
    Calls.NUMBER,
    Calls.CACHED_NAME
        };

        @Override
        public void onCreate(Bundle icicle) {
                super.onCreate(icicle);

                requestWindowFeature(Window.FEATURE_NO_TITLE);
                setContentView(R.layout.sipdroid);
                sip_uri_box2 = (AutoCompleteTextView) findViewById(R.id.txt_callee2);
                sip_uri_box2.setOnKeyListener(new OnKeyListener() {
                    public boolean onKey(View v, int keyCode, KeyEvent event) {
                        if (event.getAction() == KeyEvent.ACTION_DOWN &&
                                keyCode == KeyEvent.KEYCODE_ENTER) {
                          call_menu(sip_uri_box2);
                          return true;
                        }
                        return false;
                    }
                });
                sip_uri_box2.setOnItemClickListener(new OnItemClickListener() {
                        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                                        long arg3) {
                                call_menu(sip_uri_box2);
                        }
                });
                on(this,true);

                Button contactsButton = (Button) findViewById(R.id.contacts_button);
                contactsButton.setOnClickListener(new Button.OnClickListener() {
                        public void onClick(View v) {
                                Intent myIntent = new Intent(Intent.ACTION_DIAL);
                                startActivity(myIntent);
                        }
                });

                final Context mContext = this;
                final OnDismissListener listener = this;

                createButton = (Button) findViewById(R.id.create_button);
                createButton.setOnClickListener(new Button.OnClickListener() {
                        public void onClick(View v) {
                                CreateAccount createDialog = new CreateAccount(mContext);
                                createDialog.setOnDismissListener(listener);
                        createDialog.show();
                        }
                });

                if (!PreferenceManager.getDefaultSharedPreferences(this).getBoolean(Settings.PREF_NOPORT,
Settings.DEFAULT_NOPORT)) {
                        boolean ask = false;
                for (int i = 0; i < SipdroidEngine.LINES; i++) {
                        String j = (i!=0?""+i:"");
                        if
(PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_SERVER+j,
Settings.DEFAULT_SERVER).equals(Settings.DEFAULT_SERVER)
                                        &&
PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_USERNAME+j,
Settings.DEFAULT_USERNAME).length() != 0 &&

           PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_PORT+j,
Settings.DEFAULT_PORT).equals(Settings.DEFAULT_PORT))
                                        ask = true;
```

```
                    }
                    if (ask)
                              new AlertDialog.Builder(this)
                                        .setMessage(R.string.dialog_port)
                     .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int whichButton) {
                                        Editor edit =
PreferenceManager.getDefaultSharedPreferences(mContext).edit();
                                        for (int i = 0; i < SipdroidEngine.LINES; i++) {
                                                  String j = (i!=0?""+i:"");
                                                  if
(PreferenceManager.getDefaultSharedPreferences(mContext).getString(Settings.PREF_SERVER+j,
Settings.DEFAULT_SERVER).equals(Settings.DEFAULT_SERVER)
                                                                                      &&
PreferenceManager.getDefaultSharedPreferences(mContext).getString(Settings.PREF_USERNAME+j,
Settings.DEFAULT_USERNAME).length() != 0 &&

          PreferenceManager.getDefaultSharedPreferences(mContext).getString(Settings.PREF_PORT+j,
Settings.DEFAULT_PORT).equals(Settings.DEFAULT_PORT))
                                                            edit.putString(Settings.PREF_PORT+j, "5061");
                                        }
                                        edit.commit();
                                        Receiver.engine(mContext).halt();
                                                  Receiver.engine(mContext).StartEngine();
                                  }
                              })
                     .setNeutralButton(R.string.no, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int whichButton) {

                              }
                              })
                     .setNegativeButton(R.string.dontask, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int whichButton) {
                                        Editor edit =
PreferenceManager.getDefaultSharedPreferences(mContext).edit();
                                        edit.putBoolean(Settings.PREF_NOPORT, true);
                                        edit.commit();
                              }
                         })
                                        .show();
                    } else if (PreferenceManager.getDefaultSharedPreferences(this).getString(Settings.PREF_PREF,
Settings.DEFAULT_PREF).equals(Settings.VAL_PREF_PSTN) &&

          !PreferenceManager.getDefaultSharedPreferences(this).getBoolean(Settings.PREF_NODEFAULT,
Settings.DEFAULT_NODEFAULT))
                              new AlertDialog.Builder(this)
                                        .setMessage(R.string.dialog_default)
                     .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int whichButton) {
                                        Editor edit =
PreferenceManager.getDefaultSharedPreferences(mContext).edit();
                                        edit.putString(Settings.PREF_PREF, Settings.VAL_PREF_SIP);
                                        edit.commit();
                              }
                         })
                     .setNeutralButton(R.string.no, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int whichButton) {

                              }
                         })
                     .setNegativeButton(R.string.dontask, new DialogInterface.OnClickListener() {
                              public void onClick(DialogInterface dialog, int whichButton) {
                                        Editor edit =
PreferenceManager.getDefaultSharedPreferences(mContext).edit();
                                        edit.putBoolean(Settings.PREF_NODEFAULT, true);
                                        edit.commit();
                              }
                         })
                                        .show();
          }

          public static boolean on(Context context) {
                    return PreferenceManager.getDefaultSharedPreferences(context).getBoolean(Settings.PREF_ON,
Settings.DEFAULT_ON);
          }

          public static void on(Context context,boolean on) {
                    Editor edit = PreferenceManager.getDefaultSharedPreferences(context).edit();
                    edit.putBoolean(Settings.PREF_ON, on);
                    edit.commit();
        if (on) Receiver.engine(context).isRegistered();
          }

          @Override
          public void onResume() {
                    super.onResume();
                    if (Receiver.call_state != UserAgent.UA_STATE_IDLE) Receiver.moveTop();
                    String text;
                    text = Integer.parseInt(Build.VERSION.SDK) >= 5?CreateAccount.isPossible(this):null;
                    if (text != null && !text.contains("Google Voice") &&
                                        (Checkin.createButton == 0 || Random.nextInt(Checkin.createButton) != 0))
                              text = null;
                    if (text != null) {
                              createButton.setVisibility(View.VISIBLE);
                              createButton.setText(text);
```

```
		} else
			createButton.setVisibility(View.GONE);
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		boolean result = super.onCreateOptionsMenu(menu);

		MenuItem m = menu.add(0, ABOUT_MENU_ITEM, 0, R.string.menu_about);
		m.setIcon(android.R.drawable.ic_menu_info_details);
		m = menu.add(0, EXIT_MENU_ITEM, 0, R.string.menu_exit);
		m.setIcon(android.R.drawable.ic_menu_close_clear_cancel);
		m = menu.add(0, CONFIGURE_MENU_ITEM, 0, R.string.menu_settings);
		m.setIcon(android.R.drawable.ic_menu_preferences);

		return result;
	}

	void call_menu(AutoCompleteTextView view)
	{
		String target = view.getText().toString();
		if (m_AlertDlg != null)
		{
			m_AlertDlg.cancel();
		}
		if (target.length() == 0)
			m_AlertDlg = new AlertDialog.Builder(this)
				.setMessage(R.string.empty)
				.setTitle(R.string.app_name)
				.setIcon(R.drawable.icon22)
				.setCancelable(true)
				.show();
		else if (!Receiver.engine(this).call(target,true))
			m_AlertDlg = new AlertDialog.Builder(this)
				.setMessage(R.string.notfast)
				.setTitle(R.string.app_name)
				.setIcon(R.drawable.icon22)
				.setCancelable(true)
				.show();
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		boolean result = super.onOptionsItemSelected(item);
		Intent intent = null;

		switch (item.getItemId()) {
		case ABOUT_MENU_ITEM:
			if (m_AlertDlg != null)
			{
				m_AlertDlg.cancel();
			}
			m_AlertDlg = new AlertDialog.Builder(this)
			.setMessage(getString(R.string.about).replace("\\n","\n").replace("${VERSION}",
getVersion(this)))
			.setTitle(getString(R.string.menu_about))
			.setIcon(R.drawable.icon22)
			.setCancelable(true)
			.show();
			break;

		case EXIT_MENU_ITEM:
			on(this,false);
			Receiver.pos(true);
			Receiver.engine(this).halt();
			Receiver.mSipdroidEngine = null;
			Receiver.reRegister(0);
			stopService(new Intent(this,RegisterService.class));
			finish();
			break;

		case CONFIGURE_MENU_ITEM: {
			try {
				intent = new Intent(this, org.sipdroid.sipua.ui.Settings.class);
				startActivity(intent);
			} catch (ActivityNotFoundException e) {
			}
		}
			break;
		}

		return result;
	}

	public static String getVersion() {
		return getVersion(Receiver.mContext);
	}

	public static String getVersion(Context context) {
		final String unknown = "Unknown";

		if (context == null) {
			return unknown;
		}
```

```
                    try {
                    String ret = context.getPackageManager()
                                .getPackageInfo(context.getPackageName(), 0)
                                .versionName;
                    if (ret.contains(" + "))
                              ret = ret.substring(0,ret.indexOf(" + "))+"b";
                    return ret;
                    } catch(NameNotFoundException ex) {}

                    return unknown;
          }


          public void onDismiss(DialogInterface dialog) {
                    onResume();
          }
}
```

SipRingtonePreference.java

```
package org.sipdroid.sipua.ui;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences.Editor;
import android.media.RingtoneManager;
import android.net.Uri;
import android.preference.PreferenceManager;
import android.preference.RingtonePreference;
import android.provider.Settings;
import android.text.TextUtils;
import android.util.AttributeSet;

public class SipRingtonePreference extends RingtonePreference
{
          private Context mContext;

    public SipRingtonePreference(Context context, AttributeSet attrs)
    {
        super(context, attrs);
          mContext = context;
    }

    @Override
    protected void onPrepareRingtonePickerIntent(Intent ringtonePickerIntent)
    {
        super.onPrepareRingtonePickerIntent(ringtonePickerIntent);
        ringtonePickerIntent.putExtra(RingtoneManager.EXTRA_RINGTONE_SHOW_DEFAULT, true);
        ringtonePickerIntent.putExtra(RingtoneManager.EXTRA_RINGTONE_SHOW_SILENT, true);
        ringtonePickerIntent.putExtras(new Intent( RingtoneManager.ACTION_RINGTONE_PICKER));
    }

    @Override
    protected void onSaveRingtone(Uri ringtoneUri)
    {
        Editor edit = PreferenceManager.getDefaultSharedPreferences(mContext).edit();
                    edit.putString(org.sipdroid.sipua.ui.Settings.PREF_SIPRINGTONE, ringtoneUri != null ?
ringtoneUri.toString() : org.sipdroid.sipua.ui.Settings.DEFAULT_SIPRINGTONE);
                    edit.commit();
    }

    @Override
    protected Uri onRestoreRingtone()
    {
        String uriString =
PreferenceManager.getDefaultSharedPreferences(mContext).getString(org.sipdroid.sipua.ui.Settings.PREF_SIPRINGTONE,
                    Settings.System.DEFAULT_RINGTONE_URI.toString());
        return !TextUtils.isEmpty(uriString) ? Uri.parse(uriString) : null;
    }
}
```

- **Folder Res**

- **/Sipdroid/res/layout**

Sipdroid.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:orientation="vertical"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          android:background="#FF9900">

          <LinearLayout
                    android:orientation="horizontal"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content">

                    <LinearLayout
                              android:orientation="horizontal"
                              android:layout_weight="1"
```

```
                        android:gravity="left"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content">

                        <LinearLayout
                                  android:orientation="vertical"
                                  android:layout_width="fill_parent"
                                  android:layout_height="wrap_content"
                                  android:paddingLeft="15dip"
                                  android:paddingRight="15dip"
                                  android:paddingTop="15dip">

                                  <TextView
                                      android:layout_width="228dp"
                                      android:layout_height="64dp"
                                      android:shadowColor="#aaaaaa"
                                      android:shadowDx="1"
                                      android:shadowDy="1"
                                      android:shadowRadius="1"
                                      android:text="@string/app_name"
                                      android:textColor="@color/dtmf_dialer_background"
                                      android:textSize="28dip"
                                      android:textStyle="bold|italic"
                                      android:typeface="serif" />

                                  </LinearLayout>
            </LinearLayout>

            <LinearLayout
                        android:orientation="horizontal"
                        android:layout_weight="0"
                        android:gravity="right"
                        android:paddingTop="10dip"
                        android:paddingLeft="10dip"
                        android:paddingRight="10dip"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content">

                        <Button
                                  android:id="@+id/contacts_button"
                                  android:drawableLeft="@drawable/ic_menu_dial_pad"
                                  android:layout_width="wrap_content"
                                  android:layout_height="wrap_content"/>
            </LinearLayout>
</LinearLayout>

<LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:paddingLeft="5dip"
            android:paddingRight="5dip">

            <LinearLayout
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:padding="5dip" >

                <org.sipdroid.sipua.ui.InstantAutoCompleteTextView
                    android:id="@+id/txt_callee2"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1.87"
                    android:hint="@string/hint2"
                    android:imeOptions="actionSend"
                    android:inputType="phone"
                    android:singleLine="true"
                    android:text="" />

                <TextView
                    android:id="@+id/textView1"
                    android:layout_width="154dp"
                    android:layout_height="wrap_content"
                    android:text="RICHARD (0922001)"
                    android:textAppearance="?android:attr/textAppearanceLarge"
                    android:textColor="@color/dtmf_dialer_background" />

            </LinearLayout>

</LinearLayout>

<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FF9900"
    android:gravity="bottom"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="match_parent"
        android:layout_height="275dp"
        android:layout_weight="0.88"
        android:src="@drawable/asterisk" />
```

```
                <Button
                    android:id="@+id/create_button"
                    android:layout_width="wrap_content"
                    android:layout_height="2dp"
                    android:layout_gravity="center"
                    android:text="@string/menu_create" />
            </LinearLayout>

</LinearLayout>
```

Incall.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
android:id="@+id/top_lvl_layout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#000000"
xmlns:android="http://schemas.android.com/apk/res/android"
>

  <FrameLayout android:id="@+id/mainFrame"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_weight="1"
      android:paddingTop="10dip"
      android:paddingLeft="6dip"
      android:paddingRight="6dip"
      >

      <!-- (1) inCallPanel: the main set of in-call UI elements -->
      <RelativeLayout android:id="@+id/inCallPanel"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                >

        <!-- Slide hints: if the "sliding card" feature is enabled,
            one or the other of these is visible at any given moment
            (see updateCardSlideHints()). -->

        <!-- Slide hint and arrow *above* the main body of the card,
            shown when the card is in the *bottom* position. -->
        <!-- This hint's position onscreen is static: the Y value is set
            so that the hint will be visible just above the top edge of
            the CallCard when the CallCard is in the "bottom" position.
            The resources here describe the portrait mode layout; see
            InCallScreen.ConfigurationHelper.applyConfigurationToLayout()
            for the differences in landscape mode. -->
        <LinearLayout android:id="@+id/slideUp"
                    android:orientation="vertical"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:layout_alignParentBottom="true"
                    android:background="@null"
                    android:visibility="gone"
                    >
          <TextView android:id="@+id/slideUpHint"
                    android:layout_gravity="center_horizontal"
                    android:gravity="center_horizontal"
                    android:textAppearance="?android:attr/textAppearanceMedium"
                    android:textColor="?android:attr/textColorSecondary"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    />
          <ImageView
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_gravity="center_horizontal"
              android:src="@android:drawable/arrow_up_float"
              />
        </LinearLayout>

        <!-- Slide hint and arrow *below* the main body of the card,
            shown when the card is in the *top* position. -->
        <!-- This hint's position onscreen is static: the Y value is set
            so that the hint will be visible just below the bottom edge of
            the CallCard when the CallCard is in the "top" position.
            The resources here describe the portrait mode layout; see
            InCallScreen.ConfigurationHelper.applyConfigurationToLayout()
            for the differences in landscape mode. -->
        <LinearLayout android:id="@+id/slideDown"
                    android:orientation="vertical"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:layout_alignParentTop="true"
                     android:background="@null"
                    android:visibility="gone"
                    >
          <ImageView
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_gravity="center_horizontal"
              android:src="@android:drawable/arrow_down_float"
              />
          <TextView
```

```
            android:id="@+id/slideDownHint"
            android:layout_gravity="center_horizontal"
            android:gravity="center_horizontal"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textColor="?android:attr/textColorSecondary"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />
        <TextView
            android:id="@+id/stats"
            android:layout_gravity="center_horizontal"
            android:gravity="center_horizontal"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textColor="?android:attr/textColorSecondary"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />
        <TextView
            android:id="@+id/codec"
            android:layout_gravity="center_horizontal"
            android:gravity="center_horizontal"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textColor="?android:attr/textColorSecondary"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />
      </LinearLayout>

    </RelativeLayout>  <!-- End of inCallPanel -->

  </FrameLayout>  <!-- End of mainFrame -->

  <!-- The sliding drawer control containing the DTMF dialer.  This has been
       moved so that it is a sibling of mainFrame, instead of being a child.
       Doing so allows us to expand to the full width of the screen, instead
       of being confined to the mainFrame's layout -->
  <SlidingDrawer
    android:id="@+id/dialer_container"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"

    android:topOffset="5px"
    android:bottomOffset="7px"
    android:handle="@+id/dialer_tab"
    android:content="@+id/dtmf_dialer"
                        android:allowSingleTap="false">

    <ImageButton
      android:id="@id/dialer_tab"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:src="@drawable/ic_dialpad_tray"
      android:background="@drawable/tray_handle_normal"/>

    <include
      layout="@layout/dtmf_twelve_key_dialer"
      android:id="@id/dtmf_dialer"/>

  </SlidingDrawer>

  <ImageView
    android:id="@+id/imageView1"
    android:layout_width="294dp"
    android:layout_height="77dp"
    android:layout_x="-91dp"
    android:layout_y="386dp"
    android:src="@drawable/asterisk" />

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="8dp"
    android:layout_y="5dp"
    android:text="RICHARD" />

</AbsoluteLayout>
```

call_card.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2007 The Android Open Source Project

     Licensed under the Apache License, Version 2.0 (the "License");
     you may not use this file except in compliance with the License.
     You may obtain a copy of the License at

          http://www.apache.org/licenses/LICENSE-2.0

     Unless required by applicable law or agreed to in writing, software
     distributed under the License is distributed on an "AS IS" BASIS,
     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
     See the License for the specific language governing permissions and
     limitations under the License.
-->
```

```
<!-- XML resource file for the *children* of a CallCard used in the Phone app.
     The CallCard itself is a subclass of FrameLayout, and its (single)
     child is the LinearLayout found here.  (In the CallCard constructor,
     we inflate this file and add it as a child.)
     TODO: consider just <include>ing this directly from incall_screen.xml? -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >

    <!-- The CallCard displays info to the user about the currently active
         phone call(s) on the device.  This vertical LinearLayout contains
         the following subviews:

         (1) The "main" or "full size" call card, which displays info
             for the current foreground call, or the currently-ringing
             incoming call (if there is one.)

         (2) The "other call" info area for the current ongoing call,
             visible only if an incoming call is ringing while you're
             already using a phone line.

         (3) The "other call" info area for the current call on hold,
             visible only if there's a call on hold.
    -->

    <!-- (1) The main call card -->
    <LinearLayout
        android:id="@+id/mainCallCard"
        android:orientation="vertical"
        android:gravity="center_horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:minHeight="300dip"
        >

        <!-- "Upper title" at the very top of the CallCard. -->
        <TextView android:id="@+id/upperTitle"
                  android:paddingTop="6dip"
                  android:paddingLeft="10dip"
                  android:paddingRight="10dip"
                  android:textAppearance="?android:attr/textAppearanceLarge"
                  android:textSize="28sp"
                  android:singleLine="true"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:layout_marginTop="10dip"
                  />

        <!-- Placeholder to add some space between the title and "person info" area if
             there's enough room. -->
        <View
            android:layout_width="fill_parent"
            android:layout_height="1dip"
            android:layout_weight="1" />

        <!-- "Person info": photo / name / number -->
        <include layout="@layout/call_card_person_info" />

        <!-- Placeholder to add some space below the name/number if there's enough room. -->
        <View
            android:layout_width="fill_parent"
            android:layout_height="1dip"
            android:layout_weight="10" />

        <!-- "Lower title" and elapsed time counter, used only in the
             "call in progress" state. -->
        <LinearLayout android:id="@+id/lowerTitleViewGroup"
                      android:orientation="horizontal"
                      android:gravity="center_vertical"
                      android:layout_width="fill_parent"
                      android:layout_height="wrap_content"
                      android:layout_margin="6dip"
                      >
            <ImageView android:id="@+id/lowerTitleIcon"
                       android:layout_width="wrap_content"
                       android:layout_height="wrap_content"
                       android:layout_marginRight="8dip"
                />
            <TextView android:id="@+id/lowerTitle"
                      android:textAppearance="?android:attr/textAppearanceMedium"
                      android:textSize="18sp"
                      android:singleLine="true"
                      android:layout_width="wrap_content"
                      android:layout_weight="1"
                      android:layout_height="wrap_content"
                      android:gravity="left"
                      />
            <Chronometer android:id="@+id/elapsedTime"
                      android:textAppearance="?android:attr/textAppearanceMedium"
                      android:textSize="18sp"
                      android:singleLine="true"
                      android:layout_width="wrap_content"
```

```
                android:layout_height="wrap_content"
                />
    </LinearLayout>

</LinearLayout>    <!-- End of (1) The main call card -->


<!-- The "other call" info area. -->
<!-- There are two possible rows of information to display here:
     (1) a one-liner with info about the "ongoing" (active) call,
         displayed only if the main body of the CallCard is showing
         an incoming call and a foreground call exists.
     (2) a one-liner with info about the call on hold,
         if there's a call on hold.
     Note that BOTH can be visible in the rare case of an
     incoming call while both lines are in use.  -->
<!-- The "other call" boxes are always bottom-aligned on the call card. -->

<!-- Fix the CallCard's updateState method to just fully
     update EVERYTHING in the callcard based on the current phone
     state: set the overall type of the CallCard, load up the main
     caller info area, and load up and show or hide the "other call"
     widgets as necessary.  -->

<!-- (2) Info area for the "ongoing" call -->
<LinearLayout android:id="@+id/otherCallOngoingInfoArea"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:gravity="center_vertical"
              android:layout_marginTop="4dip"
              >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="6dip"
        android:layout_marginRight="8dip"
        android:src="@drawable/ic_incall_ongoing"
        />
    <TextView android:id="@+id/otherCallOngoingName"
          android:textAppearance="?android:attr/textAppearanceMedium"
          android:textSize="18sp"
          android:singleLine="true"
          android:layout_width="wrap_content"
          android:layout_weight="1"
          android:layout_height="wrap_content"
          android:gravity="left"
          />
    <TextView android:id="@+id/otherCallOngoingStatus"
          android:text="@string/ongoing"
          android:textAppearance="?android:attr/textAppearanceMedium"
          android:textSize="18sp"
          android:singleLine="true"
          android:layout_marginRight="6dip"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          />
</LinearLayout>

<!-- (3) Info area for the "on hold" call -->
<LinearLayout android:id="@+id/otherCallOnHoldInfoArea"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:gravity="center_vertical"
              android:layout_marginTop="4dip"
              >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="6dip"
        android:layout_marginRight="8dip"
        android:src="@drawable/ic_incall_onhold"
        />
    <TextView android:id="@+id/otherCallOnHoldName"
          android:textAppearance="?android:attr/textAppearanceMedium"
          android:textSize="18sp"
          android:singleLine="true"
          android:layout_width="wrap_content"
          android:layout_weight="1"
          android:layout_height="wrap_content"
          android:gravity="left"
          />
    <TextView android:id="@+id/otherCallOnHoldStatus"
          android:text="@string/onHold"
          android:textAppearance="?android:attr/textAppearanceMedium"
          android:textSize="18sp"
          android:singleLine="true"
          android:layout_marginRight="6dip"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          />
</LinearLayout>

<!-- The hint about the Menu button, anchored to the bottom of the
```

```
         CallCard.
         This is used only in portrait mode.  (See updateMenuButtonHint());
         in landscape mode we use the menuButtonHint from
         incall_screen.xml, which is anchored to the bottom of the
         screen.) -->
    <TextView android:id="@+id/menuButtonHint"
              android:textAppearance="?android:attr/textAppearanceMedium"
              android:textSize="18sp"
              android:textColor="?android:attr/textColorSecondary"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_gravity="center_horizontal"
              android:layout_marginTop="4dip"
              android:visibility="gone"
              android:gravity="center"
              />

</LinearLayout>
```

➢ /Sipdroid/res/layout-finger dialpad.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2006 The Android Open Source Project

     Licensed under the Apache License, Version 2.0 (the "License");
     you may not use this file except in compliance with the License.
     You may obtain a copy of the License at

          http://www.apache.org/licenses/LICENSE-2.0

     Unless required by applicable law or agreed to in writing, software
     distributed under the License is distributed on an "AS IS" BASIS,
     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
     See the License for the specific language governing permissions and
     limitations under the License.
-->

<org.sipdroid.sipua.phone.ButtonGridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/dialpad"
    android:paddingLeft="10px"
    android:paddingRight="10px"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>
        <ImageButton android:id="@+id/one"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_1_no_vm"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/two"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_2"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/three"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_3"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/four"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_4"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/five"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_5"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/six"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_6"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/seven"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_7"
            android:background="@drawable/btn_dial"
        />
```

```
        <ImageButton android:id="@+id/eight"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_8"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/nine"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_9"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/star"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_star"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/zero"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_0"
            android:background="@drawable/btn_dial"
        />

        <ImageButton android:id="@+id/pound"
            android:layout_width="96px"
            android:layout_height="76px"
            android:src="@drawable/dial_num_pound"
            android:background="@drawable/btn_dial"
        />
</org.sipdroid.sipua.phone.ButtonGridLayout>
```

dtmf_display.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/top"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:layout_marginTop="1dip">

    <!-- Number Display Field, padded for correct text alignment -->
    <EditText android:id="@+id/digits"
        android:layout_width="fill_parent"
        android:layout_height="66px"
        android:layout_marginTop="3px"
        android:layout_marginBottom="5px"
        android:layout_marginLeft="3px"
        android:layout_marginRight="3px"
        android:paddingRight="16px"
        android:paddingLeft="16px"
        android:maxLines="1"
        android:scrollHorizontally="true"
        android:textSize="28sp"
        android:freezesText="true"
        android:background="@drawable/btn_dial_textfield_normal_full"
        android:textColor="#FFFFFF"
        android:focusableInTouchMode="false"
        android:clickable="false"/>

</RelativeLayout>
```

➢ String.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--
 * Copyright (C) 2009 The Sipdroid Open Source Project
 *
 * This file is part of Sipdroid (http://www.sipdroid.org)
 *
 * Sipdroid is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * This source code is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this source code; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

////////////////////////////////////////////////////////////////////
// these are the main messages of Sipdroid
// for modifying them additional terms according to section 7, GPL apply
// see ADDITIONAL_TERMS.txt
```

```
////////////////////////////////////////////////////////////////////
-->

<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">

<string name="app_name">Asterisk Call</string>
<string name="pstn_name">Phone Call</string>
<string name="sip_name">Sipdroid Call, when available</string>

<string name="about">Sipdroid is an open-source SIP client for Android
\n
\nSee http://sipdroid.org for more info
\n
\nVersion ${VERSION}
\nThe software has been released for testing.
\n
\nCopyright (C) 2009 The Sipdroid Open Source Project (http://sipdroid.org)
\nCopyright (C) 2008 Hughes Systique Corporation, USA (http://hsc.com)
\nCopyright (C) 2006 The Android Open Source Project (http://android.com)
\nCopyright (C) 2005 Luca Veltri - University of Parma - Italy (http://mjsip.org)
\n
\nThis program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later
version.
\n
\nThis program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details
(http://www.gnu.org/licenses).</string>

<string name="reg">Registering...</string>
<string name="regpref">Preferred</string>
<string name="regclick">Click to select</string>
<string name="regfailed">Registration failed</string>

<string name="voicemail">Message(s) Waiting</string>

<string name="auto_enabled">Auto-Answer/Speakerphone</string>
<string name="auto_disabled">Auto-Answer off</string>
<string name="hint">Called Party Address</string>
<string name="hint2">Phone Number</string>
<string name="empty">Please enter called party address as name@domain.com</string>
<string name="notfast">Belum Tersambung, aktifkan Wi-Fi</string>

<string name="settings_account">SIP Account</string>
<string name="settings_line">Line</string>
<string name="settings_username">Authorization Username</string>
<string name="settings_password">Password</string>
<string name="settings_server">Server or Proxy</string>
<string name="settings_domain">Domain</string>
<string name="settings_domain2">Leave empty if same as server</string>
<string name="settings_username1">Username or Caller ID</string>
<string name="settings_callerid2">Leave empty if same as authorization username</string>
<string name="settings_port">Port</string>
          <string-array name="port_display_values">
                    <item>53</item>
                    <item>69</item>
                    <item>80</item>
                    <item>135</item>
                    <item>161</item>
                    <item>443</item>
                    <item>500</item>
                    <item>1433</item>
                    <item>1701</item>
                    <item>1812</item>
                    <item>3389</item>
                    <item>4500</item>
                    <item>5060 (standard)</item>
                    <item>5061</item>
                    <item>5900</item>
                    <item>16999</item>
                    <item>26999</item>
                    <item>36999</item>
                    <item>custom</item>
          </string-array>
          <string-array name="port_values">
                    <item>53</item>
                    <item>69</item>
                    <item>80</item>
                    <item>135</item>
                    <item>161</item>
                    <item>443</item>
                    <item>500</item>
                    <item>1433</item>
                    <item>1701</item>
                    <item>1812</item>
                    <item>3389</item>
                    <item>4500</item>
                    <item>5060</item>
                    <item>5061</item>
                    <item>5900</item>
                    <item>16999</item>
                    <item>26999</item>
                    <item>36999</item>
                    <item>0</item>
          </string-array>
```

```
<string name="settings_protocol">Protocol</string>
    <string-array name="protocol_display_values">
        <item>UDP (less standby on 3G)</item>
        <item>TCP (longer standby on 3G)</item>
    </string-array>
    <string-array name="protocol_values">
        <item>udp</item>
        <item>tcp</item>
    </string-array>
<string name="settings_vquality">Video Quality</string>
    <string-array name="vquality_display_values">
        <item>High (352x288 @ 360kbit)</item>
        <item>Low (176x144 @ 192kbit)</item>
    </string-array>
    <string-array name="vquality_values">
        <item>high</item>
        <item>low</item>
    </string-array>
<string name="settings_stun">Use STUN Server</string>
<string name="settings_stun_server">STUN Server name</string>
<string name="settings_stun_server_port">STUN Server port</string>

<string name="settings_mmtel">Use MMTel flavor</string>
<string name="settings_mmtel_qvalue">Q value</string>

<string name="settings_options">Call Options</string>
<string name="settings_wlan">Use WLAN</string>
<string name="settings_3g">Use 3G</string>
<string name="settings_edge">Use EDGE</string>
<string name="settings_vpn">Use VPN</string>
<string name="settings_check">Check with your mobile operator if he allows VoIP</string>
<string name="settings_vpn2">PBXes offers free VPN service, see PBXes >> HELP</string>

<string name="dialog_default">Sipdroid is not currently set as default phone. Would you like to make it as default
phone?
\n
\nIf you answer \'No\' here, you can still dial over Sipdroid from the system by adding \'+\' behind a number in Dialer,
or by selecting \'text to\' in Contacts.</string>
<string name="dialog_message">Would you like to see a message from Pascal introducing Sipdroid 1.5?</string>
<string name="dialog_port">Would you like to change SIP port from 5060 to 5061 for improved reliability?</string>
<string name="yes">Yes</string>
<string name="no">No</string>
<string name="dontask">Don\'t ask me again</string>

<string name="settings_pref">Preferred Call Type</string>
<string name="settings_pref2">Override by adding \'+\' behind a number in Dialer, or by selecting \'text to\' in
Contacts</string>
    <string-array name="pref_display_values">
        <item>Sipdroid only</item>
        <item>Sipdroid, when available</item>
        <item>Phone</item>
        <item>Always ask</item>
    </string-array>
    <string-array name="pref_values">
        <item>SIPONLY</item>
        <item>SIP</item>
        <item>PSTN</item>
        <item>ASK</item>
    </string-array>
<string name="settings_compression">Voice Compression</string>
    <string-array name="compression_display_values">
        <item>Only over WLAN</item>
        <item>Only over WLAN and 3G</item>
        <item>Always try</item>
        <item>Never</item>
    </string-array>
    <string-array name="compression_values">
        <item>wlan</item>
        <item>wlanor3g</item>
        <item>always</item>
        <item>never</item>
    </string-array>
<string name="settings_auto_on">Auto-Answer in use</string>
<string name="settings_auto_on2">Short vibrating alert when screen is on</string>
<string name="settings_auto_ondemand">Auto-Answer on demand</string>
<string name="settings_auto_ondemand2">Show toggle switch in status bar</string>
<string name="settings_auto_headset">Auto-Answer headset</string>
<string name="settings_auto_headset2">When wired headset is connected</string>
<string name="settings_callrecord">Record calls</string>
<string name="settings_callrecord2">Record all voice calls to SD card</string>

<string name="settings_notifications">Notifications</string>
<string name="settings_MWI">Voicemail</string>
<string name="settings_MWI2">Subscribe to SIP message waiting</string>
<string name="settings_reg">Registration</string>
<string name="settings_reg2">Allows selection of preferred SIP account</string>
<string name="settings_notify">Missed Call</string>
<string name="settings_notify2">Notify with blue LED</string>
<string name="settings_nodata">No Data</string>
<string name="settings_nodata2">Indicate missing voice packets by audible tones</string>

<string name="settings_advanced_options">Advanced Options</string>
<string name="settings_av_options">Audio/Video</string>
<string name="settings_policy_never">Changed system sleep policy to keep Wi-Fi on.</string>
```

```
<string name="settings_policy_default">Reset Wi-Fi sleep policy to default.</string>
<string name="settings_search">Search &amp;amp; Replace</string>
<string name="settings_excludepat">Exclude Pattern</string>
<string name="settings_sipringtone">SIP Ringtone</string>
<string name="settings_sipringtone2">Set ringtone for incoming SIP call</string>
<string name="settings_sipringtone_dialog">Ringtones</string>
<string name="settings_eargain">Earpiece Gain</string>
<string name="settings_micgain">Microphone Gain</string>
<string name="settings_heargain">Headset Gain</string>
<string name="settings_hmicgain">Headset Mic</string>
    <string-array name="eargain_display_values">
        <item>Low (no echo)</item>
        <item>Medium</item>
        <item>High (may cause echo to remote party)</item>
        <item>Highest</item>
    </string-array>
    <string-array name="eargain_values">
        <item>0.1</item>
        <item>0.25</item>
        <item>0.5</item>
        <item>1.0</item>
    </string-array>

<string name="settings_pbxes_options">PBXes Features</string>
<string name="settings_par">Simultaneous Outbound</string>
<string name="settings_par2">Dials all of one person\'s phone numbers in parallel</string>
<string name="settings_improve">Improve Audio</string>
<string name="settings_improve2">requires paid account</string>
<string name="settings_hdvoice">PBXes supports HD Voice</string>
<string name="settings_posurl">URL for Location/Callback</string>
<string name="settings_pos">Update Location</string>
<string name="settings_pos2">Can be announced to PBXes contacts (requires premium account)</string>
<string name="settings_pos3">Updating location</string>
<string name="settings_callback">Trigger Callback</string>
<string name="settings_callback2">If no suitable data network available</string>
<string name="settings_callthru">Trigger Callthru</string>
<string name="settings_callthru2">Prefix for Callthru</string>

<string name="settings_title">Sipdroid settings</string>

<string name="settings_profile_title">Sipdroid Settings (<xliff:g id="profile">%s</xliff:g>)</string>
<string name="settings_profile_menu_import">Import</string>
<string name="settings_profile_menu_export">Export</string>
<string name="settings_profile_menu_delete">Delete</string>
<string name="settings_profile_dialog_profiles_title">Profiles</string>
<string name="settings_profile_dialog_delete_title">Delete profile</string>
<string name="settings_profile_dialog_delete_text">Do you really want to delete <xliff:g id="profile">%s</xliff:g>
profile?</string>
<string name="settings_profile_export_error">Can not export settings!</string>
<string name="settings_profile_import_error">Can not import settings!</string>
<string name="settings_profile_delete_error">Can not delete this profile!</string>
<string name="settings_profile_delete_confirmation">Profile has been deleted.</string>
<string name="settings_wireless_options">Wireless</string>
<string name="settings_bluetooth">Requires Android 2.2 or higher</string>
<string name="settings_keepon">Screen on</string>
<string name="settings_keepon2">Required by some devices for trouble-free Wi-Fi</string>
<string name="settings_selectwifi">Select Wi-Fi AP</string>
<string name="settings_selectwifi2">By signal level</string>
<string name="settings_ownwifi">Control Wi-Fi Power</string>
<string name="settings_ownwifi2">Saves battery</string>
<string name="menu_settings">Settings</string>
<string name="menu_exit">Exit</string>
<string name="menu_about">About</string>
<string name="menu_dtmf">Send DTMF</string>
<string name="menu_video">Send Video</string>
<string name="menu_call">Dial</string>
<string name="menu_mute">Mute</string>
<string name="menu_hold">Hold</string>
<string name="menu_transfer">Transfer</string>
<string name="menu_speaker">Speaker</string>
<string name="menu_endCall">End call</string>
<string name="menu_answer">Answer</string>
<string name="menu_bluetooth">Bluetooth (experimental)</string>
<string name="transfer_title">Transfer Call to</string>
<string name="card_title_dialing">Dialing</string>
<string name="card_title_in_progress">Call in progress</string>
<string name="card_title_incoming_call">Incoming call</string>
<string name="card_title_call_ended">Call ended</string>
<string name="card_title_ended_no_codec">ERROR: Codecs Incompatible</string>
<string name="card_title_on_hold">On hold</string>
<string name="unknown">Unknown</string>
<string name="slide_hint_down_to_end_call">Geser Kebawah untuk end call.</string>
<string name="slide_hint_up_to_answer">Geser ke atas untuk menjawab.</string>
<string name="ongoing">Current Call</string>
<string name="onHold">On hold</string>
<string name="codecs">Audio Codecs</string>
<string name="codecs_summary">Select Audio Codecs to be available for voice calls</string>
<string name="codecs_move">Move this codec</string>
<string name="codecs_move_up">Move up one place</string>
<string name="codecs_move_down">Move down one place</string>

<string name="help_speakerphone">Press and hold volume buttons to force talk</string>
<string name="menu_create">New PBX linked to my Google Voice</string>

</resources>
```