”EasyAccess for Android”
Implementation Documentation

Table of Contents:

1.0 Introduction

This is the technical documentation for the implementation of EasyAccess. The document is structured as follow –

- We first describe how Generic Accessibility Guidelines such as giving feedback to user using Text to Speech or using vibratory feedback has been implemented.
- The common classes section describes the key functionality implemented in each of the classes which are used throughout the application for implementing various desired functionality of the application.
- Each app is described separately for its various components. For each of the functionality there is proper text description what we are trying to do from our code followed by how our code has been implemented. Entire function of the key functionality has been included to make sure we don’t miss out on the exact implementation of the functionality.

2.0 Generic Accessibility Considerations

2.1 Text read aloud when any item receives focus in the application

The following method attaches onFocusChangeListener to the instance of the TextView passed as a parameter. When the TextView receives focus, the content description associated with the TextView is read out and the device is made to vibrate for 300 milliseconds.

```
public void attachListener(final TextView textView) {
        textView.setOnFocusChangeListener(new OnFocusChangeListener() {
                @Override
                public void onFocusChange(View view, boolean hasFocus) {
                        if (hasFocus) {
                        giveFeedback(textView.getContentDescription().toString());
                        }
                }
        });
}
```

```
public void giveFeedback(String text) {
        // vibrate
        Vibrator vibrator = (Vibrator)
        getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(300);
                /* text to speech output code goes here */
}
```

The following method attaches onFocusChangeListener to the instance of the Button passed as a parameter. When the Button receives focus, the text on the Button is read out and the device is made to vibrate for 300 milliseconds.

```
void attachListener(Button button) {
        final String text = button.getText().toString();
        button.setOnFocusChangeListener(new OnFocusChangeListener() {
                @Override
                public void onFocusChange(View view, boolean hasFocus) {
                        if (hasFocus) {
                                giveFeedback(text);
                        }
                }
        });
}
```

The following method attaches onFocusChangeListener to the Spinner passed as parameter to the method. When the Spinner receives focus, the content description associated with the spinner is read out and the device is made to vibrate for 300 milliseconds.

```
void attachListenerToSpinner(Spinner spinner) {
        final String text = spinner.getContentDescription().toString();
        spinner.setOnFocusChangeListener(new OnFocusChangeListener() {
                @Override
                public void onFocusChange(View view, boolean hasFocus) {
                        if (hasFocus) {
                                giveFeedback(text);
                        }
                }
        });
}
```

EasyAccess Activity consists of dispatchKeyEvent method that navigates to the previous screen when the user presses the backspace key on the keyboard, and navigates to the home screen when F1 key is pressed.

```
public boolean dispatchKeyEvent(KeyEvent event) {
        if (event.getKeyCode() == KeyEvent.KEYCODE_DEL) {// go to the previous

        // screen
                // check if keyboard is connected and accessibility services are
                // disabled
                if (!Utils.isAccessibilityEnabled(getApplicationContext())
                                && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                        TTS.speak("Back");
                finish();
        } else if (event.getKeyCode() == KeyEvent.KEYCODE_F1) {// go to the home

                // screen
                // check if keyboard is connected and accessibility services are
                // disabled
                if (!Utils.isAccessibilityEnabled(getApplicationContext())
                                && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                        TTS.speak("Home");
                finish();
                Intent intent = new Intent(getApplicationContext(), SwipingUtils.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                intent.addFlags(Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
                startActivity(intent);
        }
        return super.dispatchKeyEvent(event);
}
```

The TTS output is given by default if an accessibility service is enabled. Therefore we execute the function to speak the text only if an accessibility service is not enabled and keyboard is connected to the device.

```
if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
getResources().getConfiguration().keyboard != Configuration.KEYBOARD_NOKEYS) {
        /* give text-to-speech feedback */
}
```

onKeyListener is used to execute the code when a view such as a button is selected using the Enter key on the keyboard or the center key on the Dpad.

```
button.setOnKeyListener(new OnKeyListener() {
@Override
        public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
                if(keyEvent.getAction() == KeyEvent.ACTION_DOWN) {
                        switch(keyCode) {
                                case KeyEvent.KEYCODE_DPAD_CENTER:
                                case KeyEvent.KEYCODE_ENTER:
                                        //operation to be performed when button is selected
                                        break;
                        }
                }
                return false;
        }
});
```

In case of an EditText, when the user types in the EditText, the typed character is read out. Also, if a punctuation is entered, the string before the punctuation is read out.

```
void onTextChanged(CharSequence cs, int arg1, int arg2, int arg3) {
   if(cs.length() > 0) {
        //check if keyboard is connected but accessibility services are disabled
        if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS) {
                if(cs.toString().substring(cs.length()-1,
cs.length()).matches("(?![@',&])\\p{Punct}")) {
              if(editText.getText().toString().matches("-?\\d+(\\.\\d+)?")) {
                        TTS.readNumber(editText.getText().toString());
              }
                        else {
                        TTS.speak(editText.getText().toString());
                        }
                }
                else {
                        TTS.speak(cs.toString().substring(cs.length()-1, cs.length()));
                }
        }
}
```

If a character is deleted, the deleted character is read out, and the remainder of the string is read out. If the EditText consists of digits, each digit is read separately. If the user presses the backspace key when the focus is on the EditText and no text is entered in the EditText, the current activity is destroyed, that is, the functionality of the Back button is implemented.

```
boolean dispatchKeyEvent(KeyEvent event) {
        if (event.getKeyCode() == KeyEvent.KEYCODE_DEL) {
                if (editText.getText().toString().length() != 0) {
                        /*
                         * check if keyboard is connected and accessibility services are
                         * disabled
                         */
                        if (!Utils.isAccessibilityEnabled(getApplicationContext())
                                        && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS) {
                                if (editText
                                                .getText()
                                                .toString()
                                                .substring(editText.getText().toString().length() -
1,

        editText.getText().toString().length())
                                                .matches("-?\\d+(\\.\\d+)?")) {
                                        TTS.speak("Deleted "
                                                        + editText
                                                                        .getText()
                                                                        .toString()
                                                                        .substring(

        editText.getText().toString()
```

```
    .length() - 1,

    editText.getText().toString()

    .length())
                                               + ". "
                                               + TTS.readNumber(editText
                                                          .getText()
                                                          .toString()
                                                          .substring(
                                                                    0,

    editText.getText().toString()

    .length() - 1)));
                   } else {
                        TTS.speak("Deleted "
                                               + editText
                                                          .getText()
                                                          .toString()
                                                          .substring(

    editText.getText().toString()

    .length() - 1,

    editText.getText().toString()

    .length())
                                               + ". "
                                               + editText
                                                          .getText()
                                                          .toString()
                                                          .substring(
                                                                    0,

    editText.getText().toString()
```

```
        .length() - 1));
                        }
                    }
                    editText.setText(editText.getText().toString()
                            .substring(0, editText.getText().toString().length() - 1));
                    editText.setContentDescription(editText.getText().toString()
                            .replaceAll(".(?=[0-9])", "$0 "));
                    editText.setSelection(inputContacts.getText().toString().length(),
                            inputContacts.getText().toString().length());
                    return false;
                } else {
                    // check if keyboard is connected and accessibility services are
                    // disabled
                    if (!Utils.isAccessibilityEnabled(getApplicationContext())
                            && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                        TTS.speak("Back");
                    finish();
                }
        }
}
```

2.2 Appropriate text description for all elements

```
E.g. android:contentDescription=”@string/save”
```

In strings.xml

```
<string name=”save”>Save</string>
```

2.3 Keyboard navigation of all items using arrow keys, F1 & backspace keys

```
public boolean onKeyDown(View view, Editable text, int keyCode,
                KeyEvent keyEvent) {
        switch (keyCode) {
        case KeyEvent.KEYCODE_DEL:// go to the previous screen
                // check if keyboard is connected and accessibility services are
                // disabled
                if (!Utils
                                .isAccessibilityEnabled(getActivity().getApplicationContext())
                                && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                        TTS.speak("Back");
                getActivity().finish();
                break;
        case KeyEvent.KEYCODE_F1:// go to the home screen
                // check if keyboard is connected and accessibility services are
                // disabled
                if (!Utils
                                .isAccessibilityEnabled(getActivity().getApplicationContext())
                                && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                        TTS.speak("Home");
                getActivity().finish();
                Intent intent = new Intent(getActivity().getApplicationContext(),
                                SwipingUtils.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                intent.addFlags(Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
                startActivity(intent);
                break;
        case KeyEvent.KEYCODE_DPAD_CENTER:
        case KeyEvent.KEYCODE_ENTER:
                // perform the operation to be performed when the item or view is
                // selected.
                break;
        }
        return false;
}
```

In order to implement traversal of the list/spinner items using the keyboard, the onKeyListener is used as follows:

```
listView.setOnKeyListener(new OnKeyListener() {
        @Override
        public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
                if(keyEvent.getAction() == KeyEvent.ACTION_DOWN) {
                        switch(keyCode) {
                                case KeyEvent.KEYCODE_DPAD_CENTER:
                        case KeyEvent.KEYCODE_ENTER:
                                        //action to be performed if item at currentSelection is
selected
                                        break;
                                case KeyEvent.KEYCODE_DPAD_DOWN:
                                        currentSelection++;
                                        if(currentSelection == listView.getCount()) {//end of list
                                                currentSelection = 0;
                                        }
                                        else {
                        giveFeedback(listView.getItemAtPosition(currentSelection).toString());
                                                listView.setSelection(currentSelection);
                                        }
                                        break;
                                case KeyEvent.KEYCODE_DPAD_UP:
                                        currentSelection--;
                                        if(currentSelection == -1) {//beginning of list
                                                currentSelection = listView.getCount() - 1;
                                        }
                                        else {
                        giveFeedback(listView.getItemAtPosition(currentSelection).toString());
                                                listView.setSelection(currentSelection);
                                        }
                                        break;
                        }
                }
                return false;
        }
});
```

2.4 For input boxes, describe entered text as well as modified text

2.5 Feedback to user using

Feedback to user using vibration as well as text to speech as per Accessibility service status

3.0 Common classes

This section describes the common classes whose functions are used throughout the application for various purposes.

3.1 Text To Speech

TTS class consists of methods that would be used to give a text to speech feedback to the user. It consists of a static tts object that is used to give text-to-speech feedback to the user.

```
private static TextToSpeech tts;
```

speak method reads aloud the string passed as parameter, using the tts object.

```
void speak(String message) {
        if (TTS.tts != null) {
                try {
                        TTS.tts.speak(message, TextToSpeech.QUEUE_FLUSH, null);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

stop method stops the current feedback.

```
void stop() {
        if (TTS.tts != null) {
                TTS.tts.stop();
        }
}
```

isSpeaking method returns true if the app is currently speaking any text, and false otherwise.

```
boolean isSpeaking() {
        if(tts != null) {
                return TTS.tts.isSpeaking();
        }
        return false;
}
```

readNumber method returns the characters that constitute the string passed to it, separated by a space. This can be used to read a number from left to right, one digit at a time.

```
String readNumber(String number) {
        // call a method to split the number into digits and return an ArrayList
        ArrayList<String> digits = splitNumber(number);
        String listDigits = "";
        for (String digit : digits) {
                listDigits += " " + digit;
        }
        return listDigits;
}
```

splitNumber method splits the string passed to it into characters and returns an ArrayList that consists of the characters.

```
ArrayList<String> splitNumber(String number) {
        ArrayList<String> digits = new ArrayList<String>();
        for (int i = 0; i < number.length(); i++) {
                digits.add(number.substring(i, i + 1));
        }
        return digits;
}
```

3.2 Utils class

Utils class consists of constants and common methods that would be used by the classes in EasyAccess.

applyFontTypeChanges method calls the iterateToApplyFontType method and passes the font type selected by the user.

```
void applyFontTypeChanges(Context context, LinearLayout layout) {
        // get the values in SharedPreferences
```

```
	SharedPreferences preferences = context.getSharedPreferences(context
				.getResources().getString(R.string.fonttype), 0);
	if (preferences.getInt("typeface", -1) != -1) {
		Utils.iterateToApplyFontType(layout, preferences.getInt("typeface", -1));
	} else {
		Utils.iterateToApplyFontType(layout, 0);
	}
}
```

applyFontSizeChanges and applyFontColorChanges follow the same pattern.

```
void applyFontSizeChanges(Context context, LinearLayout layout) {
	// get the values in SharedPreferences
	SharedPreferences preferences = context.getSharedPreferences(context
				.getResources().getString(R.string.size), 0);
	if (preferences.getFloat("size", 0) != 0) {
		float fontSize = preferences.getFloat("size", 0);
		Utils.iterateToApplyFontSize(layout, fontSize);
	} else {
		Utils.iterateToApplyFontSize(layout, context.getResources()
					.getDimension(R.dimen.card_textsize_regular));
		Utils.iterateToApplyFontSize(
					layout,
					Integer.valueOf(context.getResources().getString(
							R.string.defaultFontSize)));
	}
}
```

```
void applyFontColorChanges(Context context, LinearLayout layout) {
	// get the values in SharedPreferences
	SharedPreferences preferences = context.getSharedPreferences(context
				.getResources().getString(R.string.color), 0);
	if (preferences.getInt("bgcolor", -1) != -1
				|| preferences.getInt("fgcolor", -1) != -1) {
		int bgColor = preferences.getInt("bgcolor", 0);
		int fgColor = preferences.getInt("fgcolor", 0);
		try {
			context.getResources().getResourceName(bgColor);
```

```
                bgColor = context.getResources().getColor(bgColor);
            } catch (NotFoundException nfe) {
                bgColor = context.getResources().getColor(
                        R.color.card_background_regular);
            }
            try {
                context.getResources().getResourceName(fgColor);
                fgColor = context.getResources().getColor(fgColor);
            } catch (NotFoundException nfe) {
                fgColor = context.getResources().getColor(
                        R.color.card_textcolor_regular);
            }
            Utils.iterateToApplyColor(layout, bgColor, fgColor);
        } else {
            Utils.iterateToApplyColor(layout,
                    context.getResources()
                            .getColor(R.color.card_background_regular),
context
                            .getResources()
                            .getColor(R.color.card_textcolor_regular));
        }
}
```

iterateToApplyFontType method accepts a View and the font type as parameters. If the View is an instance ViewGroup, the child elements are determined and the font type of the elements that are instances of Button, TextView or RadioButton is set to the type passed as a parameter.

```
void iterateToApplyFontType(View v, int fontType) {
        if (v instanceof ViewGroup) {
            for (int index = 0; index < ((ViewGroup) v).getChildCount(); index++)
                iterateToApplyFontType(((ViewGroup) v).getChildAt(index), fontType);
        } else {
            if (v.getClass() == Button.class && v.getId() != R.id.btnNavigationBack
                    && v.getId() != R.id.btnNavigationHome) {
                switch (fontType) {
                case NONE:
                    ((Button) v).setTypeface(null, Typeface.BOLD);
                    break;
                case SERIF:
                    ((Button) v).setTypeface(Typeface.SERIF);
                    break;
                case MONOSPACE:
                    ((Button) v).setTypeface(Typeface.MONOSPACE);
                    break;
                }
            } else if (v.getClass() == TextView.class
                    && !(((TextView) v).getText().toString().trim().equals(""))) {
                switch (fontType) {
                case NONE:
                    ((TextView) v).setTypeface(null, Typeface.NORMAL);
                    break;
                case SERIF:
                    ((TextView) v).setTypeface(Typeface.SERIF);
                    break;
                case MONOSPACE:
                    ((TextView) v).setTypeface(Typeface.MONOSPACE);
                    break;
                }
            } else if (v.getClass() == RadioButton.class
                    && !(((RadioButton) v).getText().toString().trim().equals(""))) {
                switch (fontType) {
                case NONE:
                    ((RadioButton) v).setTypeface(null, Typeface.NORMAL);
                    break;
                case SERIF:
                    ((RadioButton) v).setTypeface(Typeface.SERIF);
                    break;
                case MONOSPACE:
                    ((RadioButton) v).setTypeface(Typeface.MONOSPACE);
                    break;
                }
            }
        }
}
```

iterateToApplyFontSize and iterateToApplyFontColor follow the same process.

```
void iterateToApplyFontSize(View v, float fontSize) {
        if (v instanceof ViewGroup) {
                for (int index = 0; index < ((ViewGroup) v).getChildCount(); index++)
                        iterateToApplyFontSize(((ViewGroup) v).getChildAt(index), fontSize);
        } else {
                if (v.getClass() == Button.class && v.getId() != R.id.btnNavigationBack
                                && v.getId() != R.id.btnNavigationHome) {
                        ((Button) v).setTextSize(fontSize);
                } else if (v.getClass() == TextView.class
                                && !(((TextView) v).getText().toString().trim().equals(""))) {
                        ((TextView) v).setTextSize(fontSize);
                } else if (v.getClass() == RadioButton.class
                                && !(((RadioButton) v).getText().toString().trim().equals(""))) {
                        ((RadioButton) v).setTextSize(fontSize);
                }
        }
}
```

```
void iterateToApplyColor(View view, int bgColor, int fgColor) {
        if (view instanceof ViewGroup) {
                for (int index = 0; index < ((ViewGroup) view).getChildCount(); index++)
                        iterateToApplyColor(((ViewGroup) view).getChildAt(index), bgColor,
                                        fgColor);
        } else {
                if (view.getClass() == Button.class
                                && view.getId() != R.id.btnNavigationBack
                                && view.getId() != R.id.btnNavigationHome) {
                        if (bgColor != view.getContext().getResources()
                                        .getColor(R.color.card_background_regular)) {
                                ((Button) view).setBackgroundColor(bgColor);
                        } else {
                                ((Button) view).setBackgroundDrawable(((Button) view)
                                                .getContext().

                                                getResources().getDrawable(R.drawable.card));
                        }
```

```
                              ((Button) view).setTextColor(fgColor);
                    } else if (view.getClass() == TextView.class
                                        && !(((TextView) view).getText().toString().trim().equals(""))) {
                              if (bgColor != view.getContext().getResources()
                                                  .getColor(R.color.card_background_regular)) {
                                        ((TextView) view).setBackgroundColor(bgColor);
                              } else {
                                        ((TextView)
view).setBackgroundResource(Color.TRANSPARENT);
                              }
                              ((TextView) view).setTextColor(fgColor);
                    } else if (view.getClass() == RadioButton.class
                                        && !(((RadioButton) view).getText().toString().trim()
                                                            .equals(""))) {
                              if (bgColor != view.getContext().getResources()
                                                  .getColor(R.color.card_background_regular)) {
                                        ((RadioButton) view).setBackgroundColor(bgColor);
                              } else {
                                        ((RadioButton)
view).setBackgroundResource(Color.TRANSPARENT);
                              }
                              ((RadioButton) view).setTextColor(fgColor);
                    }
          }
}
```

giveFeedback method causes the device to vibrate for 300 milliseconds, and reads aloud the string passed as a parameter.

```
 void giveFeedback(Context context, String text) {
          // vibrate
          Vibrator vibrator = (Vibrator) context
                              .getSystemService(Context.VIBRATOR_SERVICE);
          vibrator.vibrate(300);
          // TTS feedback
          if (!TTS.isSpeaking())
                    TTS.speak(text);
}
```

3.3 LayoutParamsAndViewUtils

This class stores view and the parameters to be applied to the layout.

3.4 Log

This class displays log messages, based on the log level selected.

```
void d(String msg) {
        if (logLevel == DEBUG) {
                android.util.Log.d(tag, msg);
        }
}
```

```
void setLogTag(String t) {
        tag = t;
}
```

3.5 SwipingUtils

This class adds the HomeScreenActivity to the list of fragments and loads the fragment.

```
List<Fragment> getFragments(){
        List<Fragment> fList = new ArrayList<Fragment>();
        fList.add(new HomescreenActivity());
   return fList;
}

List<Fragment> fragments = getFragments();
pageAdapter = new MyPageAdapter(getSupportFragmentManager(), fragments);
ViewPager pager = (ViewPager)findViewById(R.id.viewpager);
```

```
private class MyPageAdapter extends FragmentPagerAdapter {
        private List<Fragment> fragments;

        public MyPageAdapter(FragmentManager fm, List<Fragment> fragments) {
                super(fm);
                this.fragments = fragments;
        }
        @Override
        public Fragment getItem(int position) {
                return this.fragments.get(position);
        }
}
```

3.6 Home Screen Activity

The home screen activity displays the six options:

- Phone Dialer
- Call Log
- Text Messages
- Contacts
- Status
- Settings

startNewActivity method takes as parameter the activity that should be launched.

```
void startNewActivity(Class className) {
        Intent intent = new Intent(getActivity().getApplicationContext(), className);
        startActivity(intent);
}
```

3.7 Splash Activity

Splash activity is the first activity displayed on the screen. When this activity is created, the CallStateService is initiated.

```
Intent bootIntent = new Intent(getApplicationContext(), CallStateService.class);
bootIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startService(bootIntent);
```

The presence of a TTS engine is checked and the object in TTS class is set accordingly.

```
Intent checkIntent = new Intent();
checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent, 0);
```

```
void onActivityResult(int requestCode, int resultCode, Intent data) {
	if (requestCode == 0) {
		if (resultCode == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
			// success
			try {
				tts = new TextToSpeech(getApplicationContext(), this);
				TTS.setObject(tts);
			} catch (Exception e) {
				e.printStackTrace();
			}
		} else {
			// install the TTS data
			Intent installIntent = new Intent();
			installIntent

	.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
			startActivity(installIntent);
		}
	}
}
```

```
public void onInit(int status) {
	if (status == TextToSpeech.ERROR) {
		Toast.makeText(getApplicationContext(),
				this.getResources().getString(R.string.ttsError),
				Toast.LENGTH_LONG).show();
	}
}
```

IntentLauncher class displays the splash screen for a few seconds and redirects the user to SwipingUtils, which displays the Homescreen.

```
private class IntentLauncher extends Thread {
	@Override
	public void run() {
		try {
			// Sleeping
			Thread.sleep(SLEEP_TIME);
			//Start main activity
			Intent intent = new Intent(SplashActivity.this, SwipingUtils.class);
```

```
                        SplashActivity.this.startActivity(intent);
                        SplashActivity.this.finish();
                } catch (Exception e) {
                        // Display the error in the LogCat window
                        Log.e(TAG, e.getMessage());
                        // ImageView so that TalkBack can read it out loud
                        ImageView imageView = (ImageView) findViewById(R.id.splash);

        imageView.setContentDescription(getString(R.string.txtEasyAccessActivated));
                }
        }
}
```

3.8 EasyAccess Activity

All the activities in the application extend EasyAccess Activity. When either the back button or the home button are pressed, if screen curtain is turned on, it is turned off.

```
void turnOffScreenCurtain() {
        WindowManager windowManager = getWindowManager();
        ScreenCurtainFunctions appState = ((ScreenCurtainFunctions)
getApplicationContext());
        if (appState.getState()) {
                windowManager.removeView(curtainView);
                curtainSet = false;
                appState.setState(false);
        }
}
```

3.9 Common Adapter

CommonAdapter is a custom array adapter that displays items in a ListView.

The text size, text color and background color are retrieved from SharedPreferences and applied to the TextView.

```
        SharedPreferences preferences = context.getSharedPreferences(context
                        .getResources().getString(R.string.color), 0);
```

```
if (preferences.getInt("bgcolor", 0) != 0
                || preferences.getInt("fgcolor", 0) != 0) {
        int bgColor = preferences.getInt("bgcolor", 0);
        int fgColor = preferences.getInt("fgcolor", 0);
        try {
                context.getResources().getResourceName(bgColor);
                bgColor = context.getResources().getColor(bgColor);
        } catch (NotFoundException nfe) {
                bgColor = context.getResources().getColor(
                                R.color.homescreen_background);
        }
        try {
                context.getResources().getResourceName(fgColor);
                fgColor = context.getResources().getColor(fgColor);
        } catch (NotFoundException nfe) {
                fgColor = context.getResources().getColor(
                                R.color.card_textcolor_regular);
        }
        textView.setTextColor(fgColor);
        textView.setBackgroundColor(bgColor);
}
preferences = context.getSharedPreferences(context.getResources()
                .getString(R.string.fonttype), 0);
if (preferences.getInt("typeface", -1) != -1) {
        switch (preferences.getInt("typeface", -1)) {
        case Utils.NONE:
                textView.setTypeface(null, Typeface.NORMAL);
                break;
        case Utils.SERIF:
                textView.setTypeface(Typeface.SERIF);
                break;
        case Utils.MONOSPACE:
                textView.setTypeface(Typeface.MONOSPACE);
                break;
        }
} else {
        textView.setTypeface(null, Typeface.NORMAL);
}
```

```
        preferences = context.getSharedPreferences(context.getResources()
                        .getString(R.string.size), 0);
        if (preferences.getFloat("size", 0) != 0) {
                float fontSize = preferences.getFloat("size", 0);
                textView.setTextSize(fontSize);
        } else {
                textView.setTextSize(Integer.valueOf(context.getResources().getString(
                                R.string.textSize)));
        }
```

The content description is modified based on the content. E.g. if the content consists of time or numbers, a space is inserted between the digits and symbols such as ":", so that the text to speech service reads the content properly.

```
textView.setContentDescription(values.get(position).replaceAll(".(?=[:])","$0 "));
textView.setContentDescription(values.get(position).replaceAll(".(?=[0-9])", "$0 "));
```

4.0 Phone Dialer

The user can use the phone dialer to make and receive calls.

The user is presented with a screen that consists of buttons to dial a digit, delete the digit entered, and to make a call.

When the Phone Dialer activity is launched, the service named CallStateService is started.

4.1 CallStateService

This service listens to incoming and outgoing calls, and any change in the call state.

In case of an incoming call, the details of the caller are retrieved from the ContactManager class and the CallingScreen activity is launched that will display the details of the caller on the screen.

In case of an outgoing call, the details of the number being called are saved in Utils class, so that it can be retrieved by the other classes.

```
this.bReceiver = new BroadcastReceiver() {
@Override
        public void onReceive(Context context, Intent intent) {
```

```
                    if (intent.getAction().equals(Utils.INCOMING_CALL)) {
                                        cxt = context;
                                        String number = intent.getStringExtra("message");
                                        callingDetails = new ContactManager(getBaseContext())
                                                        .getNameFromNumber(number);
                                        // play ringtone
                                        // get custom ringtone
                                        playRingtone(number);
                                        // announce number
                                        // announceCaller(callingDetails, number);
                                        // Display Calling Activity in order to receive key events
                                        Utils.callingDetails = callingDetails;
                                        intent = new Intent(getBaseContext(),
CallingScreen.class);
                                        intent.putExtra("type", Utils.INCOMING);
                                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                                        startActivity(intent);

                    } else if (Intent.ACTION_NEW_OUTGOING_CALL.equals(intent
                                                .getAction())) {
                                        // new outgoing call
                                        final String number = intent
                                        .getStringExtra(Intent.EXTRA_PHONE_NUMBER);
                                        callingDetails = new ContactManager(getBaseContext())
                                                        .getNameFromNumber(number);
                                        Utils.callingDetails = callingDetails;
                              }
                    }
                    };
```

playRingtone method retrieve the ringtone associated with the number passed and plays it.

```
void playRingtone(String number) {
        Uri queryUri = Uri
        .withAppendedPath(ContactsContract.PhoneLookup.CONTENT_FILTER_URI,
                                        Uri.encode(number));
        String[] columns = new String[] { ContactsContract.Contacts.CUSTOM_RINGTONE
};
```

```
        Cursor contactsCursor = getContentResolver().query(queryUri, columns, null,null,
null);
        if (contactsCursor.moveToFirst()) {
                if (contactsCursor.getString(contactsCursor
                .getColumnIndex(ContactsContract.Contacts.CUSTOM_RINGTONE)) ==
null) {                         // no custom ringtone has been set
                        Utils.ringtone = RingtoneManager.getRingtone(getBaseContext(),
                                        Settings.System.DEFAULT_RINGTONE_URI);
                        Utils.ringtone.play();
                } else {
                        Utils.ringtone = RingtoneManager
                                        .getRingtone(
                                                        getBaseContext(),
                                Uri.parse(contactsCursor.getString(contactsCursor
                .getColumnIndex(ContactsContract.Contacts.CUSTOM_RINGTONE))));
                        Utils.ringtone.play();
                }
        }
}
```

CallStateListener listens to change in the state of a call. The previous and current call state are identified.
If the previous state was idle and the current state is off hook, it indicates that a new outgoing call is being made.
If the previous state was idle and the current state is ringing, it indicates that the user is getting an incoming call.
If the previous state was off hook and the current state is idle, it indicates that the call was ended or disconnected.
If the previous state was off hook and the current state is ringing, it indicates that another call is waiting.
If the previous state was off hook and the current state is off hook, it indicates that one of the active calls was disconnected or ended.
If the previous state was ringing and the current state is off hook, it indicates that the incoming call was received by the user.
If the previous state was ringing and the current state is idle, it indicates that there was a missed call.

```
public void onCallStateChanged(int newState, String incomingNumber) {
        switch (callState) {
        case TelephonyManager.CALL_STATE_IDLE:
                if (newState == TelephonyManager.CALL_STATE_OFFHOOK) {
                        // idle to off hook: new outgoing call
                        Utils.off_hook = 1;
                        Utils.ringing = 0;
                        Intent intent = new Intent(getBaseContext(), CallingScreen.class);
                        intent.putExtra("type", Utils.OUTGOING);
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        startActivity(intent);
                } else if (newState == TelephonyManager.CALL_STATE_RINGING) {
                        // idle to ringing: new incoming call
                        Utils.ringing = 1;
                        new
CallManager(getApplicationContext()).setNumber(incomingNumber);
                        sendResult(incomingNumber, Utils.INCOMING_CALL);
                }
                break;
        case TelephonyManager.CALL_STATE_OFFHOOK:
                if (newState == TelephonyManager.CALL_STATE_IDLE) {
                        // off hook to idle: call disconnected/ended, close Calling
                        // screen
                        Utils.off_hook = 0;
                        Utils.ringing = 0;
                        sendResult(getResources().getString(R.string.call_ended),
                                        Utils.CALL_ENDED);
                } else if (newState == TelephonyManager.CALL_STATE_RINGING) {
                        // off hook to ringing: another call waiting
                        Utils.ringing = 1;
                        new
CallManager(getApplicationContext()).setNumber(incomingNumber);
                        sendResult(incomingNumber, Utils.INCOMING_CALL);
                } else if (newState == TelephonyManager.CALL_STATE_OFFHOOK) {
                        // off hook to off hook: one call disconnected/ended
                        Utils.ringing = 0;
                }
                break;
        case TelephonyManager.CALL_STATE_RINGING:
                if (newState == TelephonyManager.CALL_STATE_OFFHOOK) {
                        // ringing to off hook: call answered/received
                        Utils.off_hook = 1;
                        Utils.ringing = 0;
                        if (Utils.ringtone != null && Utils.ringtone.isPlaying()) {
                            Utils.ringtone.stop();
                        }
                } else if (newState == TelephonyManager.CALL_STATE_IDLE) {
                        // ringing to idle: missed call
                        if (Utils.ringtone.isPlaying()) {
                            Utils.ringtone.stop();
                        }
                        Utils.ringing = 0;
                        Utils.off_hook = 0;
                        sendResult(getResources().getString(R.string.call_rejected),
                                        Utils.CALL_ENDED);
                }
                break;
        }
        callState = newState;
}
```

sendResult method broadcasts the state of the call passed as parameter, along with a message, if any.

```
void sendResult(String message, String intentType) {
        Intent intent = new Intent(intentType);
        if (message != null)
                intent.putExtra("message", message);
        broadcaster.sendBroadcast(intent);
}
```

The service detects change in acceleration of the device. If there is a change in the acceleration of the device and there is an incoming call, the call is received.

```
public void onShake(float force) {
        if (Utils.ringing == 1) {
                // answer call
                Intent buttonUp = new Intent(Intent.ACTION_MEDIA_BUTTON);
```

```
                    buttonUp.putExtra(Intent.EXTRA_KEY_EVENT, new KeyEvent(
                                        KeyEvent.ACTION_UP,
KeyEvent.KEYCODE_HEADSETHOOK));
                    cxt.sendOrderedBroadcast(buttonUp,
"android.permission.CALL_PRIVILEGED");
          }
}
```

If roaming is activated, the user is informed about the same.

```
cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
networkInfo = cm.getActiveNetworkInfo();
if (networkInfo != null) {
          if (networkInfo.isRoaming()) {
                    // inform the user
          }
}
```

appendDialNumber method appends the text on the button pressed by the user to the number in the TextView that displays the number to be called. It takes as parameter, the ID of the button that was pressed, the amount of time in milliseconds for which the device should vibrate, and the digit that should be appended to the contents in the TextView.

```
void appendDialNumber(int buttonInt, final int vibrateLength,
                    final String strDialDigit) {
          Button button = (Button) findViewById(buttonInt);
          button.setOnClickListener(new View.OnClickListener() {
                    public void onClick(View v) {
                              Vibrator vibrator = (Vibrator)
getSystemService(Context.VIBRATOR_SERVICE);
                              vibrator.vibrate(vibrateLength);
                              strDialNumber = strDialNumber + strDialDigit;
                              txtDialNumber.setText(strDialNumber);
                              txtDialNumber.setContentDescription(strDialNumber.replaceAll(
                                                  ".(?=[0-9])", "$0 "));
                    }
          });
}
```

The contentDescription is set such that the digits are serparated by a space, so that when the number is read out, each digit is read out separately.

callVoiceMail method retrieves the voicemail number associated with the digit pressed by the user. The digit pressed is retrieved from the TextView that displays the number. The number associated with the voicemail is displayed in the TextView.

```
void callVoiceMail() {
        Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(800);
        telManager = (TelephonyManager) getApplicationContext().getSystemService(
                        Context.TELEPHONY_SERVICE);
        if (telManager.getVoiceMailNumber() != null) {
                txtDialNumber.setText(telManager.getVoiceMailNumber());
                txtDialNumber.setContentDescription(telManager.getVoiceMailNumber()
                                .replaceAll(".(?=[0-9])", "$0 "));
                strDialNumber = txtDialNumber.getText().toString();
        }
}
```

checkAndMakeCall method makes the call if the content in the TextView is not empty.

```
void checkAndMakeCall() {
        if (strDialNumber != null && !strDialNumber.isEmpty()) {
                Vibrator vibrator = (Vibrator)
getSystemService(Context.VIBRATOR_SERVICE);
                long pattern[] = new long[] { 0, 200, 100, 200 };
                vibrator.vibrate(pattern, -1);
                makeCall();
        }
}
```

makeCall method checks the SIM card and network state, and makes the call, that is, passes the number to the default dialer app.

```
void makeCall() {
        // check SIM card availability if network is available
```

```
	if (callManager.getSimState().equals(
					getApplicationContext().getResources()
							.getString(R.string.sim_ready))) {
		callManager.setNumber(txtDialNumber.getText().toString());

		// check Network availability
		if (callManager.getServiceState().equals(
					getApplicationContext().getResources().getString(
							R.string.state_in_service))) {
			// get details of the number
			callingDetails = contactManager.getNameFromNumber(txtDialNumber
					.getText().toString());

			// Check if there is any existing call
			if (Utils.off_hook == 1) {
				announceCall(callingDetails, 1);
			}
			// pass the details to the Calling Activity
			// make call
			announceCall(callingDetails, 0);
			Utils.callingDetails = callingDetails;
			Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:"
					+ txtDialNumber.getText()));
			startActivity(intent);
			if (getIntent().getExtras() != null
					&& getIntent().getExtras().getString("call") != null) {
				finish();
			}
		} else {
			// Inform user through TTS about the network status
			// check if keyboard is connected but accessibility services are
			// disabled
		}
	} else {
		// Inform user through TTS about the SIM/network status
		if (callManager.getSimState().equals(
				getApplicationContext().getResources().getString(
						R.string.service_unknown_reason))) {
```

```
                    // inform the user about the sim state
                } else {
                    // inform the user about the sim state
                }
        }
}
```

announceCall method announces the name of the contact or the number being called. The details of the call are passed as a HashMap. If activeCall is 1, it indicates that a call is currently active.

```
void announceCall(HashMap<String, String> details, int activeCall) {
        if (activeCall == 1) {
                if (details.get("name") != null) {
                        // check if keyboard is connected but accessibility services are
                        // disabled
                        if (!Utils.isAccessibilityEnabled(getApplicationContext())
                                        && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                TTS.speak("Putting the current call on hold and calling "
                                                + details.get("name") + " " + details.get("type"));
                        Toast.makeText(
                                        getApplicationContext(),
                                        "Putting the current call on hold and calling "
                                                        + details.get("name") + " " +
details.get("type"),
                                        Toast.LENGTH_SHORT).show();
                } else {
                        // check if keyboard is connected but accessibility services are
                        // disabled
                        if (!Utils.isAccessibilityEnabled(getApplicationContext())
                                        && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                TTS.speak("Putting the current call on hold and calling "
                                                +
TTS.readNumber(txtDialNumber.getText().toString()));
                        Toast.makeText(
                                        getApplicationContext(),
                                        "Putting the current call on hold and calling "
                                                        + txtDialNumber.getText(),
Toast.LENGTH_SHORT)
                                        .show();
                }
        } else {
                if (details.get("name") != null) {
                        // check if keyboard is connected or accessibility services are
                        // disabled
                        if (Utils.isAccessibilityEnabled(getApplicationContext())
                                        || getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                TTS.speak("Calling " + details.get("name") + " "
                                                + details.get("type"));
                        Toast.makeText(
                                        getApplicationContext(),
                                        "Calling " + details.get("name") + " "
                                                        + details.get("type"),
Toast.LENGTH_SHORT).show();
                } else {
                        Toast.makeText(getApplicationContext(),
                                        "Calling " + txtDialNumber.getText().toString(),
                                        Toast.LENGTH_SHORT).show();
                        // check if keyboard is connected or accessibility services are
                        // disabled
                        if (Utils.isAccessibilityEnabled(getApplicationContext())
                                        || getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                TTS.speak("Calling "
                                                +
TTS.readNumber(txtDialNumber.getText().toString()));
                }
        }
}
```

The following code is used to reject or end a call on key down of power button:

```
if (KeyEvent.KEYCODE_POWER == event.getKeyCode()) {
                TelephonyManager telephony = (TelephonyManager)
getApplicationContext()
                                .getSystemService(Context.TELEPHONY_SERVICE);
                try {
                        Class<?> c = Class.forName(telephony.getClass().getName());
                        Method m = c.getDeclaredMethod("getITelephony");
                        m.setAccessible(true);
                        ITelephony telephonyService = (ITelephony) m.invoke(telephony);
                        telephonyService.endCall();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return true;
        }
```

On an incoming call, if up volume button is pressed, the name or number of the incoming call is announced, otherwise the loudspeaker is activated.
If down volume button is pressed, loudspeaker is deactivated.

```
Utils.audioManager = (AudioManager) getApplicationContext()
                        .getSystemService(Context.AUDIO_SERVICE);
        Utils.audioManager.setSpeakerphoneOn(true);

        if (KeyEvent.KEYCODE_VOLUME_DOWN == event.getKeyCode()) {
                if (Utils.ringing == 0 && Utils.off_hook == 1) {
                        // deactivate loudspeaker if activated
                        if (Utils.audioManager.isSpeakerphoneOn()) {
                                Utils.audioManager.setSpeakerphoneOn(false);
                        }
                        return true;
                }
        }
```

4.2 Calling Screen

The Calling Screen activity displays the details of the current call.

In case of an incoming call, an Answer button is displayed on the screen, on the click of which the call will be received by the user.

```
answerButton.setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View view) {
                              // answer call
                              Intent buttonUp = new Intent(Intent.ACTION_MEDIA_BUTTON);
                              buttonUp.putExtra(Intent.EXTRA_KEY_EVENT, new KeyEvent(
                                                  KeyEvent.ACTION_UP,
KeyEvent.KEYCODE_HEADSETHOOK));
                              getApplicationContext().sendOrderedBroadcast(buttonUp,
                                                  "android.permission.CALL_PRIVILEGED");
                    }
          });
```

When there is no active call, ,the activity is destroyed.

```
this.bReceiver = new BroadcastReceiver() {
          @Override
          public void onReceive(Context context, Intent intent) {
                    if (intent.getAction().equals(Utils.CALL_ENDED)) {
                              if (!(Utils.off_hook == 1 || Utils.ringing == 1)) {
                                        finish();
                              }
                    }
          }
};
```

The user can press the power button to reject/end incoming/active call, respectively.

```
TelephonyManager telephony = (TelephonyManager) getApplicationContext()
                              .getSystemService(Context.TELEPHONY_SERVICE);
          try {
                    Class<?> c = Class.forName(telephony.getClass().getName());
                    Method m = c.getDeclaredMethod("getITelephony");
                    m.setAccessible(true);
                    ITelephony telephonyService = (ITelephony) m.invoke(telephony);
```

```
                telephonyService.endCall();
        } catch (Exception e) {
                e.printStackTrace();
        }
```

During an incoming call, the user can press the up volume button to listen to the caller's name (or number).

```
announceCaller(this.callerDetails);

void announceCaller(String details) {
                TTS.speak(details);
}
```

During an active call, the user can press the up volume button to activate/deactivate speakerphone.

```
Utils.audioManager = (AudioManager) getApplicationContext()
                        .getSystemService(Context.AUDIO_SERVICE);
if (Utils.audioManager.isSpeakerphoneOn() == false) {
        Utils.audioManager.setSpeakerphoneOn(true);
} else {
        // deactivate loudspeaker
        Utils.audioManager.setSpeakerphoneOn(false);
}
```

During an incoming call, the user can press the down volume button to mute the ringtone.

```
Utils.ringtone.stop();
```

During an active call, the user can press the down volume button to mute/un-mute the microphone.

```
if (Utils.audioManager.isMicrophoneMute() == true) {
        Utils.audioManager.setMicrophoneMute(false);
} else {
        Utils.audioManager.setMicrophoneMute(true);
}
```

GestureListener class of CallingScreen activity listens to double tap event, that indicates that the user wants to type a number.

```
Intent intent = new Intent(getBaseContext(), PhoneDialerApp.class);
intent.putExtra("flag", 1);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

flag = 1 indicates that the Call button should not be displayed.

The user can long press on the screen to navigate to the dialer activity to make a new call.

```
Intent intent = new Intent(getBaseContext(), PhoneDialerApp.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```

4.3 Call Manager

CallManager class consists of methods that determine the state of the SIM card and the network.

It keeps track of the change in state of the service, and also determines the state of the SIM card.

```
public void onServiceStateChanged(ServiceState serviceState) {
        super.onServiceStateChanged(serviceState);
        String phonestate = "";
        // set the state of the service - serviceState according to the value in
        // serviceState
        switch (serviceState.getState()) {
        case ServiceState.STATE_EMERGENCY_ONLY:
                if (PhoneNumberUtils.isEmergencyNumber(this.number))
                        phonestate = context.getResources().getString(
                                        R.string.state_in_service);
                else
                        phonestate = context.getResources().getString(
                                        R.string.emergency_calls_only);
                setServiceState(phonestate);
                break;
        case ServiceState.STATE_OUT_OF_SERVICE:
                phonestate = context.getResources().getString(R.string.no_service);
                setServiceState(phonestate);
                break;
        case ServiceState.STATE_POWER_OFF:
                phonestate = context.getResources().getString(R.string.power_off);
                setServiceState(phonestate);
                break;
        case ServiceState.STATE_IN_SERVICE:
                phonestate = context.getResources()
                                .getString(R.string.state_in_service);
                setServiceState(phonestate);
                break;
        default:
                phonestate = context.getResources().getString(
                                R.string.service_unknown_reason);
                setServiceState(phonestate);
        }
}
```

```
public String getSimState() {
        // get the availability of the SIM card
        telMgr = (TelephonyManager) this.context
                        .getSystemService(Context.TELEPHONY_SERVICE);
        int simState = telMgr.getSimState();
        switch (simState) {
        case TelephonyManager.SIM_STATE_ABSENT:
                this.simState = this.context.getResources().getString(
                                R.string.sim_absent);
                break;
        case TelephonyManager.SIM_STATE_NETWORK_LOCKED:
                this.simState = this.context.getResources().getString(
                                R.string.network_locked);
                break;
        case TelephonyManager.SIM_STATE_PIN_REQUIRED:
                this.simState = this.context.getResources().getString(
                                R.string.pin_required);
                break;
        case TelephonyManager.SIM_STATE_PUK_REQUIRED:
                this.simState = this.context.getResources().getString(
                                R.string.puk_required);
                break;
        case TelephonyManager.SIM_STATE_READY:
                this.simState = this.context.getResources().getString(
                                R.string.sim_ready);
                break;
        case TelephonyManager.SIM_STATE_UNKNOWN:
                this.simState = this.context.getResources().getString(
                                R.string.service_unknown_reason);
                break;
        }
        return this.simState;
}
```

4.4 BootReceiver

BootReceiver starts the CallStateService when the phone boots.

```
Intent bootIntent = new Intent(context, CallStateService.class);
bootIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startService(bootIntent);
```

4.5 Accelerometer

Accelerometer detects change in the acceleration of the device.

isSupported method returns true if an accelerometer sensor is available.

```
boolean isSupported(Context context) {
        if (supported == null) {
                if (context != null) {
                        sensorManager = (SensorManager) context
                                        .getSystemService(Context.SENSOR_SERVICE);
                        // Get all sensors in device
                List<Sensor> sensors = sensorManager
                          .getSensorList(Sensor.TYPE_ACCELEROMETER);
                supported = Boolean.valueOf(sensors.size() > 0);
            } else {
                supported = Boolean.FALSE;
            }
        }
        return supported;
}
```

startListening method registers the sensor.

```
startListening(AccelerometerListener accelerometerListener) {
        sensorManager = (SensorManager) context
                .getSystemService(Context.SENSOR_SERVICE);
        // Take all sensors in device
        List<Sensor> sensors = sensorManager
                .getSensorList(Sensor.TYPE_ACCELEROMETER);
        if (sensors.size() > 0) {
            sensor = sensors.get(0);
            // Register Accelerometer Listener
            running = sensorManager.registerListener(sensorEventListener, sensor,
                        SensorManager.SENSOR_DELAY_GAME);
            listener = accelerometerListener;
        }
}
```

startListening method configures threshold and interval and registers a listener. It takes as parameter, the callback for accelerometer events, the minimum acceleration variation for considering shaking, and the minimum interval between two shake events.

```
void startListening(AccelerometerListener accelerometerListener, int threshold,
            int interval) {
        configure(threshold, interval);
        startListening(accelerometerListener);
}

void configure(int threshold, int interval) {
        Accelerometer.threshold = threshold;
        Accelerometer.interval = interval;
}
```

SensorEventListener listens to events from the accelerometer listener. onSensorChanged method triggers the shake event based on the value of the interval, force and threshold.

```
void onSensorChanged(SensorEvent event) {
        now = event.timestamp;
        x = event.values[0];
        y = event.values[1];
        z = event.values[2];
        if (lastUpdate == 0) {
                lastUpdate = now;
                lastShake = now;
                lastX = x;
                lastY = y;
                lastZ = z;
        } else {
                timeDiff = now - lastUpdate;
                if (timeDiff > 0) {
                        force = Math.abs(x + y + z - lastX - lastY - lastZ);
                        if (Float.compare(force, threshold) > 0) {
                                if (now - lastShake >= interval) {
                                        // trigger shake event
                                        listener.onShake(force);
                                }
                                lastShake = now;
                        }
                        lastX = x;
                        lastY = y;
                        lastZ = z;
                        lastUpdate = now;
                }
                // trigger change event
                listener.onAccelerationChanged(x, y, z);
        }
}
```

5.0 Status

The Status app displays the status of the following components:

- Battery level, in percentage
- Cell signal, in terms of the number of bars
- Data connection, whether 2G, 3G, or WiFi is enabled
- Number of missed calls
- Number of unread text messages
- Number of unread emails
- Current time and date
- Time and date of the next alarm
- Location data status, whether GPS is enabled
- Bluetooth status, whether it is enabled
- Brightness, whether manual or automatic, and if it is manual, it describes whether the brightness of the screen is set to low, medium or bright.

5.1 Battery Level

In order to find out the battery level, every time the battery level is changed, the current level and scale (maximum battery level) is retrieved and the battery status to be displayed is calculated using the formula: level/scale * 100.

```
public String getBatteryLevel() {
        IntentFilter filter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
        Intent batteryStatus = getApplicationContext().registerReceiver(null,
                        filter);
        int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
        int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
        float perct = (level / (float) scale) * 100;
        return String.format("%.0f", perct);
}
```

5.2 Cell Signal

Change in signal strength is detected and an integer value indicating the signal strength is retrieved, which is mapped against a certain set of values to determine the number of bars denoting the signal strength.

```
public void listenToChangeInSignalStrength() {
        SignalStrengthListener signalStrengthListener = new SignalStrengthListener();
        TelephonyManager telephonyManager = (TelephonyManager) StatusApp.this
                        .getSystemService(Context.TELEPHONY_SERVICE);
        telephonyManager.listen(signalStrengthListener,
                        PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
}
```

```
class SignalStrengthListener extends PhoneStateListener {
        public void onSignalStrengthsChanged(SignalStrength signalStrength) {
                super.onSignalStrengthsChanged(signalStrength);
                int strength = -1;
                dbm = -1024;
                if (signalStrength.isGsm()) {
                        if (signalStrength.getGsmSignalStrength() != 99) {
                                dbm = signalStrength.getGsmSignalStrength() * 2 - 113;
                        } else {
                                dbm = signalStrength.getGsmSignalStrength();
                        }
                } else {
                        dbm = signalStrength.getCdmaDbm();
                }
                if (dbm <= -111) {
                        strength = 0;
                } else if (dbm <= -99 && dbm >= -110) {
                        strength = 1;
                } else if (dbm <= -86 && dbm >= -98) {
                        strength = 2;
                } else if (dbm <= -74 && dbm >= -85) {
                        strength = 3;
                } else if ((dbm <= -61 && dbm >= -73) || (dbm >= 60 && dbm < 99)) {
                        strength = 4;
                }
        }
}
```

In order to set the text and content description of the text view associated with signal strength, the value is retrieved from the data in the message passed as a parameter to displaySignalStrength(). The content description in this case, also specifies whether the signal is absent, poor, fair, average, or good, depending on the number of bars retrieved from data in the message, as follows:

- 0: no signal
- 1: poor signal
- 2: fair
- 3: average
- 4: good

```
public void displaySignalStrength(Message msg) {
        if (msg.getData().getString("signalStrength") != null
                        && msg.getData().getString("signalStrength").equals("0")) {
                /* msg.getData().getString("signalStrength") returns the number of bars */
                /*
                 * set the content description of the TextView to the number of bars,
                 * along with a descriptive word such as poor, good etc. based on the
                 * number of bars
                 */
        }
}
```

5.3 Data Connection

The network connection status of the device is detected, such as whether 2G or 3G data connection is enabled, or whether the device is connected to the network using WIFI. The text and content description of the text view associated with data connection status are set to the value retrieved from the data in the message. If the data in the message is null, no status is displayed.

```
public boolean isConnected() {
        ConnectivityManager cm = (ConnectivityManager) getApplicationContext()
                        .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo info = cm.getActiveNetworkInfo();
        return (info != null && info.isConnected());
}
```

```
public boolean isWifiConnected() {
        ConnectivityManager cm = (ConnectivityManager) getApplicationContext()
                        .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo info = cm.getActiveNetworkInfo();
        return (info != null && info.isConnected() && info.getType() ==
ConnectivityManager.TYPE_WIFI);
}
```

```
public boolean is3gEnabled() {
        if (((TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE))
                        .getNetworkType() >= TelephonyManager.NETWORK_TYPE_UMTS)
{
                return true;
        }
        return false;
}
```

5.4 Missed Calls

All the records of type MISSED are retrieved from the CallLog, and the count is determined.

```
public int getMissedCallCount() {
        int count;
        String[] projection = { CallLog.Calls.TYPE };
        String where = CallLog.Calls.TYPE + "=" + CallLog.Calls.MISSED_TYPE;
        Cursor cursor = getContentResolver().query(CallLog.Calls.CONTENT_URI,
                        projection, where, null, null);
        count = cursor.getCount();
        cursor.close();
        return count;
}
```

5.5 Unread Text Messages

The content provider associated with SMS is queried to retrieve messages from the inbox where ‘read’ is set to 0.

```
public int getUnreadText() {
        Uri SMS_INBOX = Uri.parse("content://sms/inbox");
        int count;
        Cursor cursor = getContentResolver().query(SMS_INBOX, null, "read =   0",
                        null, null);
        count = cursor.getCount();
        cursor.close();
        return count;
}
```

5.6 Unread Emails

The number of unread conversations in the inbox of the Google accounts configured on the device is retrieved.

```
public String getAllUnreadMails() {
        Account[] accounts = AccountManager.get(this).getAccountsByType(
                        "com.google");
        ArrayList<String> accountName = new ArrayList<String>();
        String unreadEmails = "";
        int line = 1;
        for (Account account : accounts) {
                int count = 0;
                if (!(accountName.contains(account.name))) {
                        accountName.add(account.name);
                        count += getUnreadMailsByAccount(account.name);
                        // count = -1 indicates that the number of unread eMails could
                        // not be retrieved
                        if (count > 0) {
                                if (line > 1) {
                                        // add a new line before all the eMail statements
                                        // starting from the second statement
                                        unreadEmails += "<br/>";
                                }
                                unreadEmails += count + " in " + account.name + ".";
                                line++;
                        }
                }
        }
        return unreadEmails;
}
```

```
public int getUnreadMailsByAccount(String account) {
        int unread = 0;
        Cursor cursor = null;
        try {
                cursor = getContentResolver().query(
                                GmailContract.Labels.getLabelsUri(account), null, null, null,
                                null);
        } catch (Exception e) {
                e.printStackTrace();
        }
        if (cursor != null) {
                while (cursor.moveToNext()) {
                        boolean labelInboxTrue = cursor
                                        .getString(

        cursor.getColumnIndex(GmailContract.Labels.CANONICAL_NAME))

        .equals(GmailContract.Labels.LabelCanonicalNames.CANONICAL_NAME_INBOX)
;
                        boolean labelInboxPrimaryTrue = cursor
                                        .getString(

        cursor.getColumnIndex(GmailContract.Labels.CANONICAL_NAME))

        .equals(GmailContract.Labels.LabelCanonicalNames.CANONICAL_NAME_INBOX
_CATEGORY_PRIMARY);
                        if (labelInboxTrue || labelInboxPrimaryTrue) {
                                unread = cursor
                                                .getInt(cursor

        .getColumnIndex(GmailContract.Labels.NUM_UNREAD_CONVERSATIONS));
                        }
                }
                cursor.close();
        }
        return unread;
}
```

5.7 Current Time and Date
The current date and time are retrieved in MMMM dd, yyyy format.

```
public String getCurrentDateAndTime() {
        Time today = new Time(Time.getCurrentTimezone());
        today.setToNow();
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("MMMM dd, yyyy");
        String currentDateTime = sdf.format(cal.getTime());
        return currentDateTime;
}
```

5.8 Time and Date of the Next Alarm
A string representation of the net alarm is retrieved.

```
public String getNextAlarm() {
        String nextAlarm = android.provider.Settings.System.getString(
                        getContentResolver(),
Settings.System.NEXT_ALARM_FORMATTED);
        if (nextAlarm.equals("")) {
                return null;
        }
        String time = nextAlarm.substring(nextAlarm.indexOf(" "));
        String day = nextAlarm.substring(0, nextAlarm.indexOf(" "));
        nextAlarm = time + ", " + day.toString();
        return nextAlarm;
}
```

5.9 Location Data Status
It is determined whether the device supports GPS. This module llistens to change in the GPS status of the device.

```
public boolean isGpsEnabled() {
	PackageManager packageManager = 
getApplicationContext().getPackageManager();
	if 
(packageManager.hasSystemFeature(PackageManager.FEATURE_LOCATION_GPS) == 
false)
		return false;
	LocationManager manager = (LocationManager) getApplicationContext()
			.getSystemService(Context.LOCATION_SERVICE);
	boolean gpsStatus = 
manager.isProviderEnabled(LocationManager.GPS_PROVIDER);
	return gpsStatus;
}
```

```
public void listenToChangeInGpsStatus() {
	LocationManager locManager = 
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
	LocationListener locationListener = new LocationListener() {
		public void onProviderDisabled(String s) {
			if(LocationManager.GPS_PROVIDER.equals(s)) {
				//GPS is disabled
			}
		}

	};
}
```

```
public void onProviderEnabled(String s) {
	if (LocationManager.GPS_PROVIDER.equals(s)) {
		// GPS is enabled
	}
	locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
			locationListener);
}
```

5.10 Bluetooth Status

The bluetooth adapter of the device is accessed. This module listens to change in state of bluetooth and determines whether it is enabled.

```
public void listenToChangeInBluetoothStatus() {
        receiver = new BroadcastReceiver() {
                public void onReceive(Context context, Intent intent) {
                        String action = intent.getAction();
                        if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
                                int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
                                                BluetoothAdapter.ERROR);
                                switch (state) {
                                case BluetoothAdapter.STATE_ON:
                                        // Bluetooth is on
                                        break;
                                default:
                                        // Bluetooth is off
                                        break;
                                }
                        }
                }
        };

        IntentFilter filter = new IntentFilter(
                        BluetoothAdapter.ACTION_STATE_CHANGED);
        this.registerReceiver(receiver, filter);
}
```

```
public boolean isBluetoothEnabled() {
        BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (bluetoothAdapter == null) {
                return false;
        } else {
                if (!bluetoothAdapter.isEnabled()) {
                        return false;
                }
        }
        return true;
}
```

5.11 Brightness

The brightness mode is determined. It can either be automatic or manual. If it is manual, the brightness value is checked and mapped to a string that displays whether the brightness of the device is low, medium or bright.

```
public int getBrightnessMode() {
        int curBrightnessMode = 0;
        try {
                curBrightnessMode = android.provider.Settings.System.getInt(
                                getContentResolver(),

        android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE);
        } catch (SettingNotFoundException e) {
                e.printStackTrace();
                return curBrightnessMode;
        }
        return curBrightnessMode;
}
```

- Brightness Mode 0: Manual
- Brightness Mode 1: Automatic

```
public int getBrightnessValue() {
        int curBrightnessValue = 0;
        try {
                curBrightnessValue = android.provider.Settings.System.getInt(
                                getContentResolver(),
                                android.provider.Settings.System.SCREEN_BRIGHTNESS);
        } catch (SettingNotFoundException e) {
                e.printStackTrace();
                return curBrightnessValue;
        }
        return curBrightnessValue;
}
```

- Brightness Value 30: Low
- Brightness Value 102: Medium
- Brightness Value 255: Bright

These methods are called periodically using ScheduledExecutor.

```
public void getStatusPeriodically(int seconds) {
        ScheduledExecutorService executor = Executors
                        .newSingleThreadScheduledExecutor();
        Runnable periodicTask = new Runnable() {
                public void run() {
                        /* Code to retrieve the status parameters */
                }
        };
        /**
         * Execute the set of tasks periodically according to the specified number
         * of seconds to retrieve the updated values
         **/
        executor.scheduleAtFixedRate(periodicTask, 0, seconds, TimeUnit.SECONDS);
}
```

6.0 Settings

The Settings menu in EasyAccess displays the following options:

- Screen Curtain - enables/disable
- Color Settings - allows the user to change the text and background colors
- Font Settings - allows the user to change the font size and type
- Volume Settings - the user may increase/decrease the volume using the options given on the screen
- Android Settings - redirects the user to the default Android Settings screen
- About EasyAccess - displays information about EasyAccess

6.1 Screen Curtain

This function allows the user to hide the contents on the screen. The user may tap anywhere on the screen to view the screen again.
A layout with black background that covers the screen is displayed to the user.

```
View view = inflater.inflate(R.layout.screencurtain, null, false);

view.setFocusable(false);
view.setClickable(false);
view.setKeepScreenOn(false);
view.setLongClickable(false);
view.setFocusableInTouchMode(false);

LayoutParams layoutParams = new LayoutParams();

layoutParams.height = LayoutParams.MATCH_PARENT;
layoutParams.width = LayoutParams.MATCH_PARENT;
layoutParams.flags = 280;
layoutParams.format = PixelFormat.TRANSLUCENT;
layoutParams.windowAnimations = android.R.style.Animation_Toast;
layoutParams.type = LayoutParams.TYPE_SYSTEM_OVERLAY;
layoutParams.gravity = Gravity.BOTTOM;
layoutParams.x = 0;
layoutParams.y = 0;
layoutParams.verticalWeight = 1.0F;
layoutParams.horizontalWeight = 1.0F;
layoutParams.verticalMargin = 0.0F;
layoutParams.horizontalMargin = 0.0F;
```

These parameters are then associated with the view. addScreenCurtain method is called from SettingsMenu class when Screen Curtain button is pressed by the user. The view is enabled or disabled according to the current state of the application. If the view is currently visible to the user, that is, if a black screen is visible to the user, the app state is set to its previous state, and the view is removed, and vice-versa.

```
windowManager.removeView(curtainView);
```

This line of code restores the app to its previous state.

6.2 Color Settings

The user is provided with the following choices for text color and background color:

- Black
- White
- Blue
- Cyan
- Green

- Grey
- Magenta
- red
- Yellow

The colors of the elements on all the screens across the app are changed accordingly.

When the user selects a color, the applyColor method is called. This method takes as parameters, the spinner that has received focus, and the position of the item that was selected by the user.The name of the color and its corresponding resource ID is retrieved. Depending on the spinner that has focus, the selected color is saved in the appropriate file using SharedPreferences.
The change is reflected by calling applyFontColorChanges that belongs to the class Utils.

```
void applyColor(Spinner spinner, int position) {
        String color = getResources().getStringArray(R.array.colors)[position];
        int resId = getResources().getIdentifier(color.toLowerCase(), "color",
                        SettingsColor.this.getPackageName());
        switch (spinner.getId()) {
        case R.id.fcolors: // spinner associated with text colors was passed as
                                        // parameter
                // save in SharedPreferences
                preferences = getSharedPreferences(
                                getResources().getString(R.string.color), 0);
                editor = preferences.edit();
                editor.putInt("fgcolor", resId);
                break;
        case R.id.bcolors: /*
                                        * spinner associated with background colors was
passed
                                        * as parameter
                                        */
                // save in SharedPreferences
                preferences = getSharedPreferences(
                                getResources().getString(R.string.color), 0);
                editor = preferences.edit();
                editor.putInt("bgcolor", resId);
                break;
        }
        LinearLayout layout = (LinearLayout) findViewById(R.id.settingscolor);
        Utils.applyFontColorChanges(getApplicationContext(), layout);
}

```

Here, settingscolor is the ID of the layout file that is associated with the SettingsColor activity.

```
int getColorIndex(int colorValue) {
        switch (colorValue) {
        case R.color.black:
        default:
                return 0;
        case R.color.white:
                return 1;
        case R.color.blue:
                return 2;
        case R.color.cyan:
                return 3;
        case R.color.green:
                return 4;
        case R.color.grey:
                return 5;
        case R.color.magenta:
                return 6;
        case R.color.red:
                return 7;
        case R.color.yellow:
                return 8;
        }
}
```

When the user clicks on the Reset button, the reset method is called, which sets the value of fgcolor and bgcolor in SharedPreferences to -1 and calls the applyFontColorChanges from the Utils class.

```
void reset() {
        SettingsColor.this.preferences = getSharedPreferences(getResources()
                        .getString(R.string.color), 0);
        editor = SettingsColor.this.preferences.edit();
        editor.putInt("fgcolor", -1);
        editor.putInt("bgcolor", -1);
        editor.commit();

        spinnerFg.setSelection(0);
        spinnerBg.setSelection(1);

        LinearLayout layout = (LinearLayout) findViewById(R.id.settingscolor);
        Utils.applyFontColorChanges(getApplicationContext(), layout);
}
```

6.3 Font Settings

The user may change the text size and type using the Font Settings option. The current text size is displayed on the screen. The "+" and "-"buttons are used to increase and decrease the font size, respectively, by 1 unit.
The user may choose to change the type of the text to serif or monospace by selecting the appropriate option from the spinner.
The font settings are saved in the internal memory using SharedPreferenes and the changes are reflected on all the screens that constitute EasyAccess.

When the "-" button is pressed, the font size is decreased and applied to all the screens of the app. The new size is saved in the internal memory. The user may not decrease the font size beyond a limit specified in the app. The changes are applied by calling applyFontSizechanges method of Utils class.

```
if (number - 1 >= Integer.valueOf(getResources().getString(
                        R.string.lowerLimit))) {
            txtNumber.setText(Integer.toString(number - 1));
            txtNumber.setContentDescription(Integer.toString(number - 1));
            // save in SharedPreferences
            preferences = getSharedPreferences(
                        getResources().getString(R.string.size), 0);
            editor = preferences.edit();
            editor.putFloat("size", Float.valueOf(txtNumber.getText().toString()));
```

```
                if (editor.commit()) {
                        LinearLayout layout = (LinearLayout) findViewById(R.id.settingsfont);
                        Utils.applyFontSizeChanges(getApplicationContext(), layout);
                }
        }
```

Here, settingsfont is the ID of the layout file that is associated with the SettingsFont activity.

When the “+” button is pressed, the font size is increased and applied to all the screens of the app. The new size is saved in the internal memory. The user may not increase the font size beyond a limit specified in the app.

```
if (number + 1 >= Integer.valueOf(getResources().getString(
                        R.string.lowerLimit))) {
                txtNumber.setText(Integer.toString(number + 1));
                txtNumber.setContentDescription(Integer.toString(number + 1));
                // save in SharedPreferences
                preferences = getSharedPreferences(
                                getResources().getString(R.string.size), 0);
                editor = preferences.edit();
                editor.putFloat("size", Float.valueOf(txtNumber.getText().toString()));
                if (editor.commit()) {
                        LinearLayout layout = (LinearLayout) findViewById(R.id.settingsfont);
                        Utils.applyFontSizeChanges(getApplicationContext(), layout);
                }
        }
```

When the user clicks on Reset, the value associated with the font type and size are cleared.

```
SettingsFont.this.preferences =
getSharedPreferences(getResources().getString(R.string.fonttype), 0);
editor = SettingsFont.this.preferences.edit();
editor.clear();
editor.commit();
SettingsFont.this.preferences =
getSharedPreferences(getResources().getString(R.string.size), 0);
editor = SettingsFont.this.preferences.edit();
editor.putFloat("size", Float.valueOf(getResources().getString(R.string.defaultFontSize)));
editor.commit();

LinearLayout layout = (LinearLayout) findViewById(R.id.settingsfont);
Utils.applyFontTypeChanges(getApplicationContext(), layout);
Utils.applyFontSizeChanges(getApplicationContext(), layout);
```

To change the type of text, the user may select an option from the spinner. The changes are reflected in all the screens of the app. The values are saved in the internal
on item selected of spinner.

```
String typeface = Integer.toString(position); /* position refers to the position of the selected
item */
//save in SharedPreferences
preferences = getSharedPreferences(getResources().getString(R.string.fonttype), 0);
editor = preferences.edit();
editor.putInt("typeface", position);
LinearLayout layout = (LinearLayout) findViewById(R.id.settingsfont);
Utils.applyFontTypeChanges(getApplicationContext(), layout);
```

Within applyFontTypeChanges, the value of position, stored in SharedPreferences is checked and the appropriate font type is applied. 0 indicated none, 1 indicates serif and 2 indicates monospace.

6.4 Volume Settings

The user has the option to increase or decrease the media volume. The following options are given to the user:

- Loud
- Louder
- Loudest
- Normal
- Soft
- Softer
- Softest

The volume is changed based on the option selected by the user. AudioManager class is used to set the new volume.

```
        switch (radioButtonId) {
        case R.id.radioSoftest:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 3,
                                AudioManager.FLAG_PLAY_SOUND);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
                break;
        case R.id.radioSofter:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 5,
                                AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
                break;
        case R.id.radioSoft:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 7,
                                AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
                break;
        case R.id.radioNormal:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 9,
                                AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
                break;
        case R.id.radioLoud:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 11,
                                AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
                break;
        case R.id.radioLouder:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 13,
                                AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
                break;
        case R.id.radioLoudest:
                Utils.audioManager.setStreamVolume(AudioManager.STREAM_MUSIC,
                                Utils.audioManager

        .getStreamMaxVolume(AudioManager.STREAM_MUSIC),
                                AudioManager.FLAG_REMOVE_SOUND_AND_VIBRATE);
                TTS.speak(((RadioButton) findViewById(radioButtonId)).getText()
                                .toString());
        }
```

6.5 About EasyAccess

About EasyAccess displays the list of people and organizations who have supported project EasyAccess. Text Views are used to display the information.

7.0 Contacts

The Contacts app is used for searching a contact that already exists in the phone. The user may add a contact, view the details of the contact, edit the contact, or perform operations such as calling, exporting to SD Card, importing from SD card, sending a text message, making a call to the contact, deleting the contact etc.

7.1 Contact Manager

This class consists of methods to retrieve data from the contacts uri.

getNameFromNumber method accepts as parameter the number, from which the associated name and type of number should be retrieved from contacts and returned as a HashMap.

```
public HashMap<String, String> getNameFromNumber(String number) {
        // Store name and type of number in result
        HashMap<String, String> result = new HashMap<String, String>();
        // Store the number
        result.put("number", number);
        Uri lookupUri = Uri.withAppendedPath(PhoneLookup.CONTENT_FILTER_URI,
                        Uri.encode(number));
        Cursor cursor = this.context.getContentResolver().query(
                        lookupUri,
                        new String[] { ContactsContract.Contacts.DISPLAY_NAME,
                                        PhoneLookup.TYPE }, null, null, null);
        if (cursor.moveToFirst()) {
                // Store the name of the contact
                result.put("name", cursor.getString(0));
                // Store the type of number
                result.put("type", context
                                .getString(ContactsContract.CommonDataKinds.Phone
                                                .getTypeLabelResource(cursor.getInt(1))));
        }
        if (cursor != null)
                cursor.close();
        return result;
}
```

getNamesWithNumber method searches for the number in contacts and returns a string consisting of the name, type of the number, and ID of the contact.

```
public HashMap<String, ArrayList<String>> getNamesWithNumber(String strNumber) {
        // Store name and type of number in result
        HashMap<String, ArrayList<String>> result = new HashMap<String,
ArrayList<String>>();
        ArrayList<String> name, number, type, id;
        name = new ArrayList<String>();
        number = new ArrayList<String>();
        type = new ArrayList<String>();
        id = new ArrayList<String>();
        Cursor cursor = context.getContentResolver().query(
                        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                        new String[] { Phone.DISPLAY_NAME, Phone.TYPE,
Phone.NUMBER,
                                        Phone.CONTACT_ID }, Phone.NUMBER + "    LIKE ?",
                        new String[] { strNumber + "%" }, null);
        if (cursor != null) {
                while (cursor.moveToNext()) {
                        name.add(cursor.getString(0));
                        type.add(Phone.getTypeLabel(this.context.getResources(),
                                        cursor.getInt(1), "").toString());
                        number.add(cursor.getString(2));
```

```
                    id.add(cursor.getString(3));
                }
        }
        result.put("name", name);
        result.put("number", number);
        result.put("type", type);
        result.put("id", id);
        return result;
}
```

getNamesStartingWith method searches for the names that start with the string passed, in contacts and returns a HashMap consisting of the name, type of the number and the ID of the contact.

```
public HashMap<String, ArrayList<String>> getNamesStartingWith(String strName) {
        // Store name, type of number and ID in result
        HashMap<String, ArrayList<String>> result = new HashMap<String,
ArrayList<String>>();
        ArrayList<String> name, id, type, number;
        name = new ArrayList<String>();
        id = new ArrayList<String>();
        type = new ArrayList<String>();
        number = new ArrayList<String>();
        String sortOrder = ContactsContract.Contacts.DISPLAY_NAME
                        + " COLLATE LOCALIZED ASC";
        Cursor cursor = context.getContentResolver().query(
                        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                        new String[] {
ContactsContract.CommonDataKinds.Phone.CONTACT_ID,
                                Phone.DISPLAY_NAME, Phone.TYPE,
Phone.NUMBER },
                        ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME + "
LIKE ?",
                        new String[] { strName + "%" }, sortOrder);
        while (cursor.moveToNext()) {
                id.add(cursor.getString(0));
                name.add(cursor.getString(1));
                type.add(Phone.getTypeLabel(this.context.getResources(),
                                    cursor.getInt(2), "").toString());
                number.add(cursor.getString(3));
        }
        result.put("name", name);
        result.put("type", type);
        result.put("id", id);
        result.put("number", number);
        if (cursor != null)
                cursor.close();
        return result;
}
```

getNumber method takes as a parameer, the ID of the contact, and returns the corresponding phone number.

```
public String getNumber(String idOfContact) {
            String contactNumber = "";
            Cursor cursor = context.getContentResolver().query(
                                    ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                                    new String[] { Phone.NUMBER },
                                    ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ? ",
                                    new String[] { idOfContact }, null);
            if (cursor.moveToNext()) {
                        contactNumber = cursor.getString(cursor

            .getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
            }
            return contactNumber;
}
```

getDetailsFromId method searches for the ID of the contact and returns the corresponding name, numbers, type of the numbers and emails of the contact, in a HashMap.

```
public HashMap<String, ArrayList<String>> getDetailsFromId(String id) {
            ArrayList<String> ids, name = null, number = null, type = null, email = null;
            HashMap<String, ArrayList<String>> contacts = new HashMap<String,
ArrayList<String>>();
            ids = new ArrayList<String>();
```

```
        Cursor cur = context.getContentResolver().query(
                        ContactsContract.Contacts.CONTENT_URI, null,
                        ContactsContract.Contacts._ID + " = ?", new String[] { id },
                        "display_name" + " ASC");
        if (cur.getCount() > 0) {
                while (cur.moveToNext()) {
                        name = new ArrayList<String>();
                        number = new ArrayList<String>();
                        type = new ArrayList<String>();
                        email = new ArrayList<String>();
                        String displayName = cur.getString(cur

        .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                        name.add(displayName);
                        ids.add(id);
                        Cursor pCur = context.getContentResolver()

        .query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                                                null,

        ContactsContract.CommonDataKinds.Phone.CONTACT_ID
                                                                + " = ? ", new String[] { id },
null);
                        while (pCur.moveToNext()) {
                                String phoneNumber = pCur
                                                .getString(pCur

        .getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
                                number.add(phoneNumber);
                                int numberType =
pCur.getInt(pCur.getColumnIndex(Phone.TYPE));
                                String contactNumberType = Phone.getTypeLabel(
                                                this.context.getResources(), numberType,
"").toString();
                                type.add(contactNumberType);
                        }
                        pCur.close();
                        Cursor emailCur = this.context.getContentResolver().query(
        ContactsContract.CommonDataKinds.Email.CONTENT_URI, null,

        ContactsContract.CommonDataKinds.Email.CONTACT_ID + " = ?",
                                    new String[] { id }, null);
                        while (emailCur.moveToNext()) {
                                String mail = emailCur
                                                .getString(emailCur

        .getColumnIndex(ContactsContract.CommonDataKinds.Email.ADDRESS));
                                if (email.size() == 0 || !email.contains(mail))
                                        email.add(mail);
                        }
                        emailCur.close();
                }
        }
        cur.close();
        contacts.put("name", name);
        contacts.put("numbers", number);
        contacts.put("types", type);
        contacts.put("emails", email);
        contacts.put("id", ids);
        return contacts;
}
```

getAllContacts method returns the details (name, ID and number) of all the contacts on the device in a HashMap.

```
public HashMap<String, ArrayList<String>> getAllContacts() {
        ArrayList<String> ids, name, number;
        HashMap<String, ArrayList<String>> contacts = new HashMap<String,
ArrayList<String>>();
        ids = new ArrayList<String>();
        name = new ArrayList<String>();
        number = new ArrayList<String>();

        Cursor cur = context.getContentResolver().query(
                        ContactsContract.Contacts.CONTENT_URI, null, null, null,
                        "display_name" + " ASC");
        if (cur.getCount() > 0) {
                while (cur.moveToNext()) {
                        String id = cur.getString(cur
                                        .getColumnIndex(ContactsContract.Contacts._ID));
                        String displayName = cur.getString(cur

        .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                        if (Integer
                                        .parseInt(cur.getString(cur

        .getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER))) > 0) {
                                Cursor pCur = context.getContentResolver().query(

        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                                                null,

        ContactsContract.CommonDataKinds.Phone.CONTACT_ID
                                                        + " = ?", new String[] { id }, null);
                                if (pCur.moveToFirst()) {
                                        String phoneNumber = pCur
                                                        .getString(pCur

        .getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
                                        number.add(phoneNumber);
                                        name.add(displayName);
                                        ids.add(id);
                                }
                                pCur.close();
                        }
                }
        }
        cur.close();
        contacts.put("name", name);
        contacts.put("number", number);
        contacts.put("id", ids);
        return contacts;
}
```

getId method takes as parameter the number of the contact and returns the corresponding ID.

```
public String getId(String contactNumber) {
	String contactId = null;
	ContentResolver contentResolver = context.getContentResolver();
	Uri uri = Uri.withAppendedPath(
			ContactsContract.PhoneLookup.CONTENT_FILTER_URI,
			Uri.encode(contactNumber));
	String[] projection = new String[] {
			ContactsContract.Contacts.DISPLAY_NAME, PhoneLookup._ID };
	Cursor cursor = contentResolver.query(uri, projection, null, null, null);
	if (cursor != null) {
		while (cursor.moveToNext()) {
			contactId = cursor.getString(cursor
					.getColumnIndexOrThrow(PhoneLookup._ID));
			break;
		}
		cursor.close();
	}
	return contactId;
}
```

7.2 ContactsApp

The Contacts App displays all the contacts on the device in a ListView and provides a search box where the user can type a name or a number and the list is filtered accordingly.

```
public void run() {
	file = new File(getDir("data", MODE_PRIVATE), "contacts");
	ObjectInputStream inputStream = null;
	if (file.length() != 0) {
		try {
				inputStream = new ObjectInputStream(new
FileInputStream(file));
		} catch (FileNotFoundException e) {
				e.printStackTrace();
		} catch (IOException e) {
```

```
                                e.printStackTrace();
                }
                try {
                        try {
                                ContactsApp.this.contactsMap = (HashMap<String,
ArrayList<String>>) inputStream.readObject();
                                inputStream.close();
                                // fetch contacts and display, call this thread,
                                // pass to handler
                                new Thread(new Runnable() {
                                        public void run() {
                                                name =
ContactsApp.this.contactsMap.get("name");
                                                number =
ContactsApp.this.contactsMap.get("number");
                                                idArrayList
=ContactsApp.this.contactsMap.get("id");
                                                numberArrayList = number;
                                                nameArrayList = name;
                                                contactIdArrayList = idArrayList;
                                                Bundle bundle = new Bundle();
                                                Message message = new Message();
                                                bundle.putInt("type", ALL_CONTACTS);
                                                message.setData(bundle);
                                                handler.sendMessage(message);
                                                        new Thread(new Runnable() {
                                                                public void run() {

        ContactsApp.this.contactsMap = contactManager.getAllContacts();
                                                                        Bundle bundle = new
Bundle();
                                                                        Message message = new
Message();
                                                                        bundle.putInt("type",
ALL_CONTACTS);
                                                                        message.setData(bundle);

        handler.sendMessage(message);
```

```
                                                }
                                            }).start();
                                        }
                                    }).start();
                                } catch (EOFException e) {
                                    e.printStackTrace();
                                } catch (ClassNotFoundException e) {
                                    e.printStackTrace();
                                }
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                        } else {
                            try {
                                ContactsApp.this.outputStream = new
ObjectOutputStream(new FileOutputStream(file));
                                ContactsApp.this.contactsMap =
contactManager.getAllContacts();

        ContactsApp.this.outputStream.writeObject(ContactsApp.this.contactsMap);
                                ContactsApp.this.outputStream.close();
                                name = contactsMap.get("name");
                                number = contactsMap.get("number");
                                idArrayList = contactsMap.get("id");
                                numberArrayList = number;
                                nameArrayList = name;
                                contactIdArrayList = idArrayList;
                                Bundle bundle = new Bundle();
                                Message message = new Message();
                                bundle.putInt("type", ALL_CONTACTS);
                                message.setData(bundle);
                                handler.sendMessage(message);
                            } catch (FileNotFoundException e) {
                                e.printStackTrace();
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                        }
```

```
}
}



//display the list passed to the handler
handler = new Handler() {
                        @Override
                        public void handleMessage(Message message) {
                                if (message.getData().getInt("type") == ALL_CONTACTS) {
                                        progressBar.setVisibility(View.GONE);
                                        // display the contacts in the ListView
                                        contactsListView.setVisibility(View.VISIBLE);
                                        contactsAdapter = new ContactsAdapter(
                                                        getApplicationContext(), name);
                                        contactsListView.setAdapter(contactsAdapter);
};
```

If the contacts are being fetched for the first time, they are saved in a file in the internal memory. If the contacts are already there in the internal memory, those contacts are displayed in the ListView and the contacts are again retrieved in a separate thread and these contacts are loaded in the ListView once all the contacts have been fetched, to ensure that the changes (if any), are reflected on the user's display. The newly fetched contacts then replace the contacts in the file in the internal memory. This is done in order to save time as fetching the contacts takes time. Using this method, the previously fetched contacts are displayed to the user, while at the same time, the contacts are being fetched again in a separate thread.

If a contact is selected from the list view, the ContactsDetailsMenu activity is launched. The details of the contact are passed along with the intent.

```
Intent intent = new Intent(getApplicationContext(), ContactsDetailsMenu.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
if (numberArrayList.size() > 0) {
        intent.putExtra("number", numberArrayList.get(position));
        intent.putExtra("name", nameArrayList.get(position));
        intent.putExtra("id", idArrayList.get(position));
}
```

```
startActivity(intent);
finish();
```

loadList method loads the contacts in the ListView based on the string entered by the user in the EditText.

```
        if (!(inputContacts.getText().toString().matches("-?\\d+(\\.\\d+)?"))) {
                for (int i = 0; i < name.size(); i++) {
                        if (name.get(i)
                                        .toString()
                                        .toLowerCase()
                                        .startsWith(

        inputContacts.getText().toString().toLowerCase())) {
                                nameArrayList.add(name.get(i));
                                numberArrayList.add(number.get(i));
                                idArrayList.add(contactIdArrayList.get(i));
                                contactsAdapter = new
ContactsAdapter(getApplicationContext(),
                                                nameArrayList);
                                found = 1;
                        }
                }
        } else { // user entered a number
                for (int i = 0; i < number.size(); i++) {
                        if (number.get(i).toString()
                                        .startsWith(inputContacts.getText().toString())) {
                                nameArrayList.add((new
ContactManager(getApplicationContext())

        .getNameFromNumber(number.get(i))).get("name"));
                                numberArrayList.add(number.get(i));
                                idArrayList.add(contactIdArrayList.get(i));
                                contactsAdapter = new
ContactsAdapter(getApplicationContext(),
                                                nameArrayList);
                                found = 1;
                        }
```

```
                    }
          }
```

Here, inputContacts refers to the EditText in which the user is expected to enter the string to be searched for. If the user enters a letter, then the string is considered to be a name and the name is searched in Contacts, If the user entered a digit, the phone number is searched in Contacts. The values in the ListView are modified accordingly. If the number entered by the user is not saved in contacts, two buttons, Call and Save are displayed on the screen.

```
if (found == 1) {
                    btnCall.setVisibility(View.GONE);
                    btnSave.setVisibility(View.GONE);
                    contactsListView.setAdapter(contactsAdapter);
} else {
                    // if user typed a NUMBER that is not present in Contacts, display
                    // Call and Save buttons
          if (inputContacts.getText().toString().matches("-?\\d+(\\.\\d+)?")) {
                              btnCall.setVisibility(View.VISIBLE);
                              btnSave.setVisibility(View.VISIBLE);
          }
}
```

callContact method launches the PhoneDialerApp activity and passes the name (or the number) of the contact to be called.

```
Intent intent = new Intent(getApplicationContext(),
                              PhoneDialerApp.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
intent.putExtra("call", inputText);
startActivity(intent);
finish();
```

saveContact method launches the SaveContact activity. The number that should be saved is passed with the intent.

```
Intent intent = new Intent(getApplicationContext(), SaveContact.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
intent.putExtra("number", inputText);
startActivity(intent);
finish();
```

7.3 Contact Details

The Contact Details activity lists the phone numbers and email IDs of the selected contact and provides options for operations such as calling, sending text message, modifying the selected details of the contact.

The details of the contacts are retrieved from ContactManager class and displayed in the form of buttons. When the user clicks on the button (e.g. Call Mobile, Edit Home), the corresponding operation is performed.

The code create a Call button is given below:

```
Button btnCall = new Button(getApplicationContext());
        btnCall.setText("Call " + numType + Html.fromHtml("<br/>" + num));

        btnCall.setGravity(Gravity.CENTER);
        btnCall.setFocusable(true);
        LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
                        LayoutParams.MATCH_PARENT,
LayoutParams.WRAP_CONTENT);
        btnCall.setLayoutParams(params);
        LinearLayout layout = (LinearLayout) findViewById(R.id.linearLayout);
        layout.addView(btnCall);

        btnCall.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View view) {
                        callDialer(view);
                }
        });
```

editNumber method takes as parameters, the ID of the contact, number and the type of number. The ContactUpdate activity is launched and these three parameters are passed to the activity along with the intent.

```
void editNumber(String id, String num, String numType) {
        Intent intent = new Intent(getApplicationContext(), ContactUpdate.class);
        intent.putExtra("id", id);
        intent.putExtra("number", num);
        intent.putExtra("numtype", numType);
        startActivity(intent);
        finish();
}
```

editMail method takes as parameters, the ID and the email ID of the contact. The ContactUpdate activity is launched and these three parameters are passed to the activity along with the intent.

```
void editMail(String id, String email) {
        Intent intent = new Intent(getApplicationContext(), ContactUpdate.class);
        intent.putExtra("id", id);
        intent.putExtra("mail", email);
        startActivity(intent);
        finish();
}
```

callDialer method takes as parameter, the Button that was clicked by the user. The PhoneDialerApp activity is launched and the number to be called is passed along with the intent.

```
void callDialer(View view) {
        // pass number to dialer app
        Intent intent = new Intent(getApplicationContext(), PhoneDialerApp.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        int positionOfNewLine = (((Button) view).getText().toString())
                        .indexOf("\n");
        intent.putExtra("call", (((Button) view).getText().toString()
                        .substring(positionOfNewLine)).trim());
        startActivity(intent);
        finish();
}
```

callMessagesApp method takes as parameter, the Button that was clicked by the user. The TextMessagesComposerApp activity is launched and the number to be called is passed along with the intent.

```
void callMessagesApp(View view) {
        // pass number to dialer app
        Intent intent = new Intent(getApplicationContext(),
                        TextMessagesComposerApp.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        int positionOfNewLine = (((Button) view).getText().toString())
                        .indexOf("\n");
        intent.putExtra("number", (((Button) view).getText().toString()
                        .substring(positionOfNewLine)).trim());
        startActivity(intent);
        finish();
}
```

The user can click on More Options button to view the list of operations that can be performed, other than call, send sms and send email. The ID of the contact and number are passed to ContactsOtherOptions activity.

```
void callOtherOptions() {
        Intent intent = new Intent(getApplicationContext(),
                        ContactsOtherOptions.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.putExtra("id", ContactsDetailsMenu.this.contactId);
        intent.putExtra("number", ContactsDetailsMenu.this.number);
        startActivity(intent);
        finish();
}
```

7.4 Other Options

The user is provided with the following options:

- Delete
- Copy to SD Card
- Import All Contacts from SD Card
- Export All Contacts to SD Card

7.4.1 Delete

deleteContact method takes as parameter, the number of the contact to be deleted. It returns true on successful deletion and false otherwise.

```
boolean deleteContact(String number) {
	Uri contactUri = Uri.withAppendedPath(PhoneLookup.CONTENT_FILTER_URI,
				Uri.encode(number));
	Cursor cur = getContentResolver().query(contactUri, null, null, null, null);
	try {
		if (cur.moveToFirst()) {
			do {
				String lookupKey = cur.getString(cur

	.getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));
				Uri uri = Uri
						.withAppendedPath(

	ContactsContract.Contacts.CONTENT_LOOKUP_URI,
								lookupKey);
				getContentResolver().delete(uri, null, null);
				return true;
			} while (cur.moveToNext());
		}
	} catch (Exception e) {
		e.getStackTrace();
	}
	return false;
}
```

7.4.2 Copy To SD Card

User can export the selected contact as a .vcf file to the SD card.
copyToSDcard method checks the status of the SD card and creates a .vcf file of the contact in the SD card.

```
void copyToSDcard() {
	if (getSDcardStatus()) {
		Runnable runnable = new Runnable() {
			@Override
```

```
                              public void run() {
                                        Bundle bundle = new Bundle();
                                        Message message = new Message();
                                        Cursor cursor =
getContentResolver().query(Data.CONTENT_URI,
                                                            null, Data.CONTACT_ID + " = ?",
                                                            new String[] { ContactsOtherOptions.this.id },
null);
                                        if (getVCF(cursor)) {
                                                  bundle.putBoolean("success", true);
                                        } else {
                                                  bundle.putBoolean("success", false);
                                        }
                                        bundle.putInt("type", EXPORTONECONTACT);
                                        message.setData(bundle);
                                        handler.sendMessage(message);
                              }
                    };
                    new Thread(runnable).start();
          }
```

```
boolean getSDcardStatus() {
          if (Environment.getExternalStorageState().equals(
                              Environment.MEDIA_MOUNTED)) {
                    if (Environment.getExternalStorageState().equals(
                                                  Environment.MEDIA_MOUNTED_READ_ONLY)) {
                                        //SD card is read only
                                        return false;
                              } else {
                                        return true;
                              }
                    }
          return false;
}
```

getVCF exports the selected contact to the SD card in EasyAccess directory. It takes as parameter, the cursor containing the details of the contact. It returns true if the contact was successfully exported, and returns false otherwise.

```
public boolean getVCF(Cursor cursor) {
        while (cursor.moveToNext()) {
                String lookupKey = cursor.getString(cursor
                                .getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));
                Uri uri = Uri.withAppendedPath(
                                ContactsContract.Contacts.CONTENT_VCARD_URI,
lookupKey);
                AssetFileDescriptor fd;
                try {
                        fd = getContentResolver().openAssetFileDescriptor(uri, "r");
                        FileInputStream fis = fd.createInputStream();
                        byte[] buf = new byte[(int) fd.getDeclaredLength()];
                        fis.read(buf);
                        String VCard = new String(buf);
                        String vfile = cursor.getString(

                        cursor.getColumnIndex("display_name")).replaceAll("[^a-zA-Z0-9_]",
                                        "")
                                        + "_"
                                        + cursor.getString(cursor

        .getColumnIndex(ContactsContract.Contacts._ID))
                                        + ".vcf";
                        String path = Environment.getExternalStorageDirectory().toString()
                                        + File.separator
                                        + getResources().getString(R.string.appName);
                        File directory = new File(path);
                        if (!directory.exists())
                                directory.mkdirs();
                        File outputFile = new File(path, vfile);
                        FileOutputStream mFileOutputStream = new FileOutputStream(
                                        outputFile);
                        mFileOutputStream.write(VCard.toString().getBytes());
                } catch (Exception e) {
                        e.printStackTrace();
                        return false;
                }
        }
        return true;
}
```

7.4.3 Import all contacts from SD card

importContact method reads the .vcf files from the EasyAccess directory in the SD card and writes to Contacts.

```
private void importContact() {
        final Intent intent = new Intent(Intent.ACTION_VIEW);
        final File[] files = new File(Environment.getExternalStorageDirectory()+
File.separator + getResources().getString(R.string.appName)).listFiles();
        Runnable runnable = new Runnable() {
                        @Override
                        public void run() {
                                for (int i = 0; i < files.length; i++) {
                                        if (files[i].getName().endsWith(".vcf")) {
                                                try {
                                                        intent.setDataAndType(Uri.fromFile(new
File(files[i].getAbsolutePath())),"text/x-vcard");
                                                        startActivity(intent);
                                                } catch (Exception e) {
                                                        //import failed
                                                }
                                        }
                                }
                        }
        };
        new Thread(runnable).start();
}
```

7.4.4 Export all contacts to SD card

export method exports all contacts to the SD card as .vcf files.

```
void export() {
        if (getSDcardStatus()) {
                Runnable runnable = new Runnable() {
                        @Override
                        public void run() {
                                Bundle bundle = new Bundle();
                                Message message = new Message();
                                Cursor cursor =
getContentResolver().query(Data.CONTENT_URI,
                                                null, null, null, null);
                                if (getVCF(cursor)) {
                                        bundle.putBoolean("success", true);
                                } else {
                                        bundle.putBoolean("success", false);
                                }
                                bundle.putInt("type", EXPORTALLCONTACTS);
                                message.setData(bundle);
                                handler.sendMessage(message);
                        }
                };
                new Thread(runnable).start();

        }
}
```

7.5 Saving to contacts

The number and the corresponding details - name, number, type of number, email - are saved in the device's contacts.

saveContact method stores the details in Contacts and returns true on successful addition and false otherwise.

```
boolean saveContact() {
        ArrayList<ContentProviderOperation> op_list = new
ArrayList<ContentProviderOperation>();
        op_list.add(ContentProviderOperation
                        .newInsert(ContactsContract.RawContacts.CONTENT_URI)
                        .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, null)
                        .withValue(ContactsContract.RawContacts.ACCOUNT_NAME,
null).build());

        // first and last names
        op_list.add(ContentProviderOperation
                        .newInsert(Data.CONTENT_URI)
                        .withValueBackReference(Data.RAW_CONTACT_ID, 0)
                        .withValue(Data.MIMETYPE,
StructuredName.CONTENT_ITEM_TYPE)
                        .withValue(

        ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME,
                                this.editName.getText().toString()).build());

        op_list.add(ContentProviderOperation
                        .newInsert(Data.CONTENT_URI)
                        .withValueBackReference(Data.RAW_CONTACT_ID, 0)
                        .withValue(ContactsContract.Data.MIMETYPE,

        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
                        .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER,
                                this.number)
                        .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
                                this.typeIndex)
                        .withValue(ContactsContract.CommonDataKinds.Phone.LABEL,
                                this.spinnerType.getSelectedItem().toString()).build());
        op_list.add(ContentProviderOperation
                        .newInsert(Data.CONTENT_URI)
                        .withValueBackReference(Data.RAW_CONTACT_ID, 0)
                        .withValue(ContactsContract.Data.MIMETYPE,

        ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)
                        .withValue(ContactsContract.CommonDataKinds.Email.DATA,
                                        editEmail.getText().toString()).build());

        try {
                ContentProviderResult[] results = getContentResolver().applyBatch(
                                ContactsContract.AUTHORITY, op_list);
        } catch (Exception e) {
                e.printStackTrace();
                return false;
        }
        return true;
}
```

saveAndGiveFeedback method edits the contact by calling editContact() and gives the feedback to the user if keyboard is connected and accessibility services are disabled. The value of editFlag is checked. If it is true, the contact details are updated. If it is false, the details are saved in contacts.

```
void saveAndGiveFeedback() {
        if (editFlag == true) {
                // get Id of contact
                String id = new ContactManager(getApplicationContext())
                                .getId(SaveContact.this.number);
                // edit contact
                if (editContact(id)) {
                        Toast.makeText(getApplicationContext(),
                                        getResources().getString(R.string.contactupdated),
                                        Toast.LENGTH_SHORT).show();
                        // check if keyboard is connected but accessibility services are
                        // disabled
                        if (!Utils.isAccessibilityEnabled(getApplicationContext())
                                        && getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                TTS.speak(getResources().getString(R.string.contactupdated));
                        finish();
                } else {
                        Toast.makeText(getApplicationContext(),
                                        getResources().getString(R.string.contactnotupdated),
                                        Toast.LENGTH_SHORT).show();
```

```
				// check if keyboard is connected but accessibility services are
				// disabled
				if (!Utils.isAccessibilityEnabled(getApplicationContext())
						&& getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)

	TTS.speak(getResources().getString(R.string.contactnotupdated));
		}
	} else {
		if (saveContact()) {
			Toast.makeText(getApplicationContext(),
					getResources().getString(R.string.contactsaved),
					Toast.LENGTH_SHORT).show();
			// check if keyboard is connected but accessibility services are
			// disabled
			if (!Utils.isAccessibilityEnabled(getApplicationContext())
					&& getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
				TTS.speak(getResources().getString(R.string.contactsaved));
			finish();
		} else {
			Toast.makeText(getApplicationContext(),
					getResources().getString(R.string.contactnotsaved),
					Toast.LENGTH_SHORT).show();
			// check if keyboard is connected but accessibility services are
			// disabled
			if (!Utils.isAccessibilityEnabled(getApplicationContext())
					&& getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)

	TTS.speak(getResources().getString(R.string.contactnotsaved));
		}
	}
}
```

editContact method updates the details of an existing contact. It takes as parameter, the Id of the contact. It returns true on successful updation and false on failure.

```
boolean editContact(String id) {
        ArrayList<ContentProviderOperation> op_list = new
ArrayList<ContentProviderOperation>();
        ContentResolver cr = getContentResolver();
        String where = Data.RAW_CONTACT_ID + "=?";
        String[] params = new String[] { id };
        Cursor phoneCur = getContentResolver().query(
                        ContactsContract.Data.CONTENT_URI, null, where, params, null);
        phoneCur.moveToFirst();
        ArrayList<ContentProviderOperation> ops = new
ArrayList<ContentProviderOperation>();
        // first and last names
        ops.add(ContentProviderOperation
                        .newUpdate(ContactsContract.RawContacts.CONTENT_URI)
                        .withSelection(Data._ID + "=?", new String[] { id })
                        .withValue("display_name", this.editName.getText().toString())
                        .build());

        op_list.add(ContentProviderOperation
                        .newUpdate(Data.CONTENT_URI)
                        .withSelection(Data._ID + "=?", new String[] { id })
                        .withValue(ContactsContract.Data.MIMETYPE,

        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)

                        .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER,
                                this.editNumber.getText().toString())
                        .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
                                this.typeIndex)
                        .withValue(ContactsContract.CommonDataKinds.Phone.LABEL,
                                this.spinnerType.getSelectedItem().toString()).build());
        op_list.add(ContentProviderOperation
                        .newUpdate(Data.CONTENT_URI)
                        .withSelection(Data._ID + "=?", new String[] { id })
                        .withValue(ContactsContract.Data.MIMETYPE,

        ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)
                        .withValue(ContactsContract.CommonDataKinds.Email.DATA,
                                        editEmail.getText().toString()).build());
        phoneCur.close();
        try {
                ContentProviderResult[] result = cr.applyBatch(
                                ContactsContract.AUTHORITY, ops);
                if (result[0] == null)
                        return false;
                return true;
        } catch (RemoteException e) {
                e.printStackTrace();
                return false;
        } catch (OperationApplicationException e) {
                e.printStackTrace();
                return false;
        }
}
```

7.6 Updating Contact

The attribute of the contact that is selected by the user is updated using ContactUpdate activity.

If the attribute such as phone number or mail, is set as primary already, the option to set it to primary should not be displayed.

isPrimaryNumber method can be used to check if the number passed as a parameter is the primary phone number of the contact. The method also takes as parameter, the contact ID of the contact.

```
boolean isPrimaryNumber(String contactId, String contactNumber) {
        Cursor cursor = getContentResolver().query(Phone.CONTENT_URI,
                        new String[] { Phone.DATA, Phone.IS_PRIMARY },
                        Phone.CONTACT_ID + "=?", new String[] { contactId }, null);
        int index = cursor.getColumnIndex(Phone.DATA);
        while (cursor.moveToNext()) {
                String phoneNumber = cursor.getString(index);
                if (phoneNumber.equals(contactNumber)) {
                        if (!(cursor.getString(cursor.getColumnIndex(Phone.IS_PRIMARY))
                                        .equals("0"))) {
                                // number is primary
```

```
                    cursor.close();
                    return true;
                }
            }
        }
        cursor.close();
        return false;
}
```

isPrimaryMail method can be used to check if the mail ID passed as a parameter is the primary mail ID of the contact. The method also takes as parameter, the contact ID of the contact.

```
boolean isPrimaryMail(String contactId, String emailId) {
        Cursor cursor = getContentResolver().query(
                        ContactsContract.CommonDataKinds.Email.CONTENT_URI, null,
                        ContactsContract.CommonDataKinds.Email.CONTACT_ID + " = ?",
                        new String[] { contactId }, null);
        while (cursor.moveToNext()) {
                String mail = cursor
                                .getString(cursor

        .getColumnIndex(ContactsContract.CommonDataKinds.Email.ADDRESS));
                if (emailId.equals(mail)) {
                        if (!(cursor
                                        .getString(cursor

        .getColumnIndex(ContactsContract.CommonDataKinds.Email.IS_PRIMARY))
                                        .equals("0"))) {
                                // email id is primary
                                return true;
                        }
                }
        }
        cursor.close();
        return false;
}
```

setPrimaryNumber method is used to change the primary number associated with a contact. It takes as parameters, the contact ID of the contact, and the phone number of the contact. The method returns true if the phone number is successfully set as the primary number, and false otherwise.

```
boolean setPrimaryNumber(String contactId, String contactNumber) {
        ArrayList<ContentProviderOperation> ops = new
ArrayList<ContentProviderOperation>();
        Cursor cursor = getContentResolver().query(Phone.CONTENT_URI, null,
                        Phone.CONTACT_ID + "=?", new String[] { contactId }, null);
        String where = ContactsContract.Data.CONTACT_ID + " = ? AND "
                        + ContactsContract.CommonDataKinds.Phone.NUMBER + " = ?";
        String[] params = new String[] { contactId, contactNumber };
        ops.add(ContentProviderOperation
                        .newUpdate(ContactsContract.Data.CONTENT_URI)
                        .withSelection(where, params)

        .withValue(ContactsContract.CommonDataKinds.Phone.IS_PRIMARY, 1)
                        .build());
        try {
                getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
                return true;
        } catch (RemoteException e) {
                e.printStackTrace();
        } catch (OperationApplicationException e) {
                e.printStackTrace();
        }
        return false;
}
```

setPrimaryMail method is used to change the primary email ID associated with a contact. It takes as parameters, the contact ID of the contact, and the email ID of the contact. The method returns true if the email ID is successfully set as the primary email ID, and false otherwise.

```
boolean setPrimaryMail(String contactId, String emailId) {
        ArrayList<ContentProviderOperation> ops = new
ArrayList<ContentProviderOperation>();
```

```
		Cursor cursor = getContentResolver().query(
					ContactsContract.CommonDataKinds.Email.CONTENT_URI, null,
					ContactsContract.CommonDataKinds.Email.CONTACT_ID + "=?",
					new String[] { contactId }, null);
		String where = ContactsContract.CommonDataKinds.Email.CONTACT_ID
					+ " = ? AND " +
ContactsContract.CommonDataKinds.Email.ADDRESS
					+ " = ?";
		String[] params = new String[] { contactId, emailId };
		ops.add(ContentProviderOperation.newUpdate(Data.CONTENT_URI)
					.withSelection(where, params)
					.withValue(ContactsContract.CommonDataKinds.Email.IS_PRIMARY,
1)
					.build());
		try {
			getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
			return true;
		} catch (RemoteException e) {
			e.printStackTrace();
		} catch (OperationApplicationException e) {
			e.printStackTrace();
		}
		return false;
}
```

saveContact method saves the deetails of the number in the contacts of the device. It returns true on success and false on failure.

```
boolean saveContact() {
		ArrayList<ContentProviderOperation> op_list = new
ArrayList<ContentProviderOperation>();
		op_list.add(ContentProviderOperation
					.newInsert(ContactsContract.RawContacts.CONTENT_URI)
					.withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, null)
					.withValue(ContactsContract.RawContacts.ACCOUNT_NAME,
null).build());
		// first and last names
		op_list.add(ContentProviderOperation
                              .newInsert(Data.CONTENT_URI)
                              .withValueBackReference(Data.RAW_CONTACT_ID, 0)
                              .withValue(Data.MIMETYPE,
StructuredName.CONTENT_ITEM_TYPE)
                              .withValue(

          ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME,
                                          this.editField.getText().toString()).build());
          op_list.add(ContentProviderOperation
                              .newInsert(Data.CONTENT_URI)
                              .withValueBackReference(Data.RAW_CONTACT_ID, 0)
                              .withValue(ContactsContract.Data.MIMETYPE,

          ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
                              .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER,
                                          this.number)
                              .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
                                          this.typeIndex)
                              .withValue(ContactsContract.CommonDataKinds.Phone.LABEL,
                                          this.spinnerType.getSelectedItem().toString()).build());
          try {
                    ContentProviderResult[] results = getContentResolver().applyBatch(
                              ContactsContract.AUTHORITY, op_list);
          } catch (Exception e) {
                    e.printStackTrace();
                    return false;
          }
          return true;
}
```

editContact method updates the selected attribute of the selected contact with the values entered by the user. It returns true on success and false on failure.

```
boolean editContact(String contactId, int flag) {
          ArrayList<ContentProviderOperation> ops = new
ArrayList<ContentProviderOperation>();
          switch (flag) {
```

```
        case NAME:
                ops.add(ContentProviderOperation
                                .newUpdate(Data.CONTENT_URI)
                                .withSelection(
                                                Data.CONTACT_ID + "=? AND " +
Data.MIMETYPE + "='"
                                                                +
StructuredName.CONTENT_ITEM_TYPE + "'",
                                                new String[] { contactId })
                                .withValue(StructuredName.DISPLAY_NAME,
                                                this.editField.getText().toString()).build());
                break;
        case NUMBER:
                ops.add(ContentProviderOperation
                                .newUpdate(Data.CONTENT_URI)
                                .withSelection(
                                                Data.CONTACT_ID
                                                                + "=? AND "
                                                                +
ContactsContract.Data.MIMETYPE
                                                                + "='"
                                                                +
ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
                                                                + "'", new String[] { contactId })

        .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER,
                                                this.editField.getText().toString())
                                .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
                                                this.typeIndex)

        .withValue(ContactsContract.CommonDataKinds.Phone.LABEL,

        this.spinnerType.getSelectedItem().toString()).build());
                break;
        case EMAIL:
                ops.add(ContentProviderOperation
                                .newUpdate(Data.CONTENT_URI)
                                .withSelection(
                                        Data.CONTACT_ID
                                                  + "=? AND "
                                                  +
ContactsContract.Data.MIMETYPE
                                                  + "='"
                                                  +
ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
                                                  + "'", new String[] { contactId })

     .withValue(ContactsContract.CommonDataKinds.Email.ADDRESS,
                                        this.editField.getText().toString()).build());
          break;
     }
     try {
          ContentProviderResult[] results = getContentResolver().applyBatch(
                    ContactsContract.AUTHORITY, ops);
     } catch (Exception e) {
          e.printStackTrace();
          return false;
     }
     return true;
}
```

7.7 Contacts Adapter

7.7.1 Side Selector

Side Selector enabled quick-letter navigation through the list of contacts.

init method initializes the paint object.

```
void init() {
     setBackgroundColor(0x44FFFFFF);
     paint = new Paint();
     paint.setColor(0xFFA6A9AA);
     paint.setTextSize(20);
     paint.setTextAlign(Paint.Align.CENTER);
}
```

setListView adds the section index to the ListView passed as a parameter.

```
void setListView(ListView _list) {
	list = _list;
	selectionIndexer = (SectionIndexer) _list.getAdapter();

	Object[] sectionsArr = selectionIndexer.getSections();
	sections = new String[sectionsArr.length];
	for (int i = 0; i < sectionsArr.length; i++) {
		sections[i] = sectionsArr[i].toString();
	}
}
```

Here, sections is a String array, and selectionIndexer is a SectionIndexer.

7.7.2 SpinnerAdapter and Contact Adapter

Spinner Adapter and Contact Adapter create custom adapters for adding items in the spinner and list view, respectively. The getCustomView method retrieves the text color and background color from the internal memory using SharedPreferences and associates the colors with the TextView. Similarly the text size and text types are retrieved and applied to the TextViews.

```
public View getCustomView(int position, View view, ViewGroup parent) {
	LayoutInflater inflater = (LayoutInflater) context
				.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
	View rowView = inflater.inflate(R.layout.row, parent, false);
	final TextView textView = (TextView) rowView.findViewById(R.id.textView1);
	textView.setText(values.get(position));
	textView.setContentDescription(values.get(position));
	SharedPreferences preferences = context.getSharedPreferences(context
				.getResources().getString(R.string.color), 0);
	int bgColor = preferences.getInt("bgcolor", 0);
	int fgColor = preferences.getInt("fgcolor", 0);
	try {
		if (bgColor != 0) {
```

```
                    context.getResources().getResourceName(bgColor);
                    bgColor = context.getResources().getColor(bgColor);
                    textView.setBackgroundColor(bgColor);
          } else {
                    textView.setBackgroundDrawable(null);
          }
} catch (NotFoundException nfe) {
          textView.setBackgroundDrawable(null);
}
try {
          context.getResources().getResourceName(fgColor);
          fgColor = context.getResources().getColor(fgColor);
} catch (NotFoundException nfe) {
          fgColor = context.getResources().getColor(
                              R.color.card_textcolor_regular);
}
textView.setTextColor(fgColor);

preferences = context.getSharedPreferences(context.getResources()
                    .getString(R.string.fonttype), 0);
if (preferences.getInt("typeface", -1) != -1) {
          switch (preferences.getInt("typeface", -1)) {
          case Utils.NONE:
                    textView.setTypeface(null, Typeface.NORMAL);
                    break;
          case Utils.SERIF:
                    textView.setTypeface(Typeface.SERIF);
                    break;
          case Utils.MONOSPACE:
                    textView.setTypeface(Typeface.MONOSPACE);
                    break;
          }
} else {
          textView.setTypeface(null, Typeface.NORMAL);
}

preferences = context.getSharedPreferences(context.getResources()
                    .getString(R.string.size), 0);
```

```
        if (preferences.getFloat("size", 0) != 0) {
                float fontSize = preferences.getFloat("size", 0);
                textView.setTextSize(fontSize);
        } else {
                textView.setTextSize(Integer.valueOf(context.getResources().getString(
                                R.string.textSize)));
        }

        return rowView;
}
```

8.0 Text Messaging

The Text Messaging app allows the user to compose, view and delete text messages.

8.1 SmsReceiver

The SmsReceiver class listens to incoming messages and plays the corresponding ringtone. IT then redirects to the Text Messaging app of EasyAccess. If version of the OS running on the device is greater than 18, and EasyAccess is set as the default messaging app, we have to write the code to send the message to the inbox.

```
if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")) {
                Bundle bundle = intent.getExtras();
                SmsMessage[] msgs = null;
                String sender = null;
                if (bundle != null) {
                        try {
                                Object[] pdus = (Object[]) bundle.get("pdus");
                                msgs = new SmsMessage[pdus.length];
                                // read the details of the originating address and search for
                                // the number in contacts
                                // play ringtone
                                Utils.ringtone = RingtoneManager.getRingtone(context,

        Settings.System.DEFAULT_NOTIFICATION_URI);
                                Utils.ringtone.play();
                                // if default app
                                if (android.os.Build.VERSION.SDK_INT >= 19) {
                                        if
(Telephony.Sms.getDefaultSmsPackage(context).equals(
                                                        context.getPackageName())) {
                                                ContentValues values = new ContentValues();
                                                /*
                                                 * add the details of the sender, and the details of
the
                                                 * message to values. Also set read attribute to 0
to
                                                 * indicate that the message is unread.
                                                 */
                                                context.getContentResolver().insert(
                                                            Uri.parse("content://sms/inbox"),
values);
                                        }
                                }
                                Intent intentObject = new Intent(context,
TextMessagesApp.class);
                                intentObject.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                                context.startActivity(intentObject);

                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }
```

8.2 Text Messaging Activity

When the text messaging app is loaded, the ListView is populated with the messages in the inbox. 5 messages are displayed on the screen at a time. The user may swipe left or right to view the next or previous 5 messages. To view a message the user has to click on an item in the ListView. The conversation will be loaded in the subsequent screen that comes to the foreground.

The checkIfDefault method casks the user whether hr JustDridroid ld bshoulde made the deault app for viewing text messages.
If the user selects Yes, EasyAccess is saved as the default app.

```
final String myPackageName = getPackageName();
if (android.os.Build.VERSION.SDK_INT >= 19) {
        if (!Telephony.Sms.getDefaultSmsPackage(this).equals(myPackageName)) {
                        // App is not default.
                        // Show the "not currently set as the default SMS app" dialog
                        Intent intent = new
Intent(Sms.Intents.ACTION_CHANGE_DEFAULT);
                        intent.putExtra(Sms.Intents.EXTRA_PACKAGE_NAME,
                                getApplicationContext().getPackageName());
                        startActivity(intent);
        }
}
```

The getMessages method takes as a parameter, the type of message, that is, inbox, or sent.
Based on the type of message passed, the ListView is populated with the corresponding messages.

```
switch (typeOfMessage) {
        case INBOX:
                uri = "content://sms/inbox";
                break;
        case SENT:
                uri = "content://sms/sent";
                break;
}

this.cursor = getContentResolver().query(Uri.parse(uri),
                        new String[] { "DISTINCT address", "date", "read", "subject","body" },
"address IS NOT NULL) GROUP BY (address", null,null);
```

In order to enable browsing through the items in the list using a keyboard, it is necessary to associate a key listener to the ListView. Up and down arrow keys are used to shift the focus on the items in the ListView.

```
int currentSelection = -1;

public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
        if (keyEvent.getAction() == KeyEvent.ACTION_DOWN) {
                switch (keyCode) {
                case KeyEvent.KEYCODE_DPAD_CENTER:

        startNewActivity(TextMessagesApp.this.numbers.get(currentSelection));
                        break;
                case KeyEvent.KEYCODE_DPAD_DOWN:
                        currentSelection++;
                        if (currentSelection == lstView.getCount()) {
                                currentSelection = 0;
                        }
                        break;
                case KeyEvent.KEYCODE_DPAD_UP:
                        currentSelection--;
                        if (currentSelection == -1) {
                                currentSelection = lstView.getCount() - 1;
                        } else {
                                messageListView.setSelection(currentSelection);
                        }
                        break;
                }
        }
        return false;
}
```

The onItemClickListener is used to launch the TextMessagesViewerApp activity in order to load the conversation from the sender of the message selected by the user. The number of the sender is passed along with the intent.

```
messageListView.setOnItemClickListener(new OnItemClickListener() {
@Override
        public void onItemClick(AdapterView<?> arg0, View view,
                                        int position, long arg3) {
                startNewActivity(TextMessagesApp.this.numbers.get(position));
        }
});

void startNewActivity(String number) {
			Intent intent = new Intent(getApplicationContext(),
TextMessagesViewerApp.class);
			intent.putExtra("address", number);
			startActivity(intent);
}
```

When the user clicks on the compose button, the TextMessagesComposerRecipientApp activity is launched.

```
Intent intent = new Intent(getApplicationContext(),
TextMessagesComposerRecipientApp.class);
startActivity(intent);
```

A class that extends SimpleOnGestureListener is used to detect fling actions.

```
class MyGestureDetector extends SimpleOnGestureListener {
@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
			float velocityY) {
	try {
		if (Math.abs(e1.getY() - e2.getY()) > SWIPE_MAX_OFF_PATH)
			return false;
		// right to left swipe
		if (e1.getX() - e2.getX() > SWIPE_MIN_DISTANCE
				&& Math.abs(velocityX) > SWIPE_THRESHOLD_VELOCITY) {
			// get next 5 messages
		} else if (e2.getX() - e1.getX() > SWIPE_MIN_DISTANCE
				&& Math.abs(velocityX) > SWIPE_THRESHOLD_VELOCITY) {
			// get previous 5 messages
		}
	} catch (Exception e) {
		e.printStackTrace();
	}
	return false;
}
}
```

8.3 Viewing Conversation

The messages associated with the number passed to this activity are displayed, with the latest message at the top of the list. The user may choose to delete the thread or delete an individual message. EasyAccess also provides a facility to call the sender or to send a reply.

When the user clicks on the **Call** button, the PhoneDialer App is launched and the number to be called is passed along with the intent.

```
Intent intent = new Intent(getApplicationContext(), PhoneDialerApp.class);
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
intent.putExtra("call", TextMessagesViewerApp.this.address);
startActivity(intent);
```

When the user clicks on the **Reply** button, the TextMessagesComposerApp activity is launched. If the number of the recipient exists in contacts, the name, number and type of number (mobile, home etc.) are passed along with the intent. If the number is not saved on the device, the number alone is passed with the intent.

```
Intent intent = new Intent(getApplicationContext(), TextMessagesComposerApp.class);
intent.putExtra("number", numberOfRecipient);
/* save the details of the contact in a HashMap named map, if the number is saved on the
device. */
if(map.get("name") != null) {
   intent.putExtra("name", map.get("name"));
   intent.putExtra("type", map.get("type"));
}
startActivity(intent);
```

When the user clicks on the delete button, an AlertDialog is displayed asking the user to confirm whether the message or the thread should be deleted. If the user confirms that the message or the thread should be deleted, the deleteMessage or the deleteThread method is called.
deleteMessage() accepts as parameter, the time at which the message was sent or received. The number of the sender/recipient is stored in the class. These two variables are used to delete a message.

```
boolean deleteMessage(String dateTimestamp) {
        Uri deleteUri = Uri.parse("content://sms");
        if (getContentResolver().delete(deleteUri, "address=? AND date = ?",
                        new String[] { this.address, dateTimestamp }) != 0) {
                return true;
        }
        return false;
}
```

deleteThread() uses the number stored in the class and deletes all the messages associated with the number.

```
boolean deleteThread() {
        Uri deleteUri = Uri.parse("content://sms");
        if (getContentResolver().delete(deleteUri, "address=?",
                        new String[] { this.address }) != 0) {
                return true;
        }
        return false;
}
```

Here, address is the number associated with the message to be deleted.
getMessages() retrieves all the messages associated with the number and stores the details in a HAshMap named records.

```
        String uri = "content://sms";
        cursor = getContentResolver().query(Uri.parse(uri),
                        new String[] { "subject", "body", "type", "date" }, "address = ?",
                        new String[] { address }, null);
        do {
                values = new ArrayList<String>();
                try {
                        String date = this.cursor.getString(this.cursor
                                        .getColumnIndex("date"));
                        values.add(this.cursor.getString(this.cursor
                                        .getColumnIndex("subject")));

        values.add(this.cursor.getString(this.cursor.getColumnIndex("body")));
                        if (cursor.getString(cursor.getColumnIndex("type"))
                                        .equalsIgnoreCase("1")) {
```

```
                                // sms received
                                values.add(Integer.toString(INBOX));
                        } else if (cursor.getString(cursor.getColumnIndex("type"))
                                        .equalsIgnoreCase("2")) {
                                values.add(Integer.toString(SENT));
                        }
                        records.put(date, values);
                } catch (Exception e) {
                        continue;
                }
        } while (this.cursor.moveToNext()
                        && this.cursor.getPosition() != this.cursor.getCount());
        sort(records);
```

sort() sorts the records with the latest message first and dynamically generates TextViews and Buttons to be displayed on the screen.

```
void sort(HashMap hashMap) {
        Map<Integer, String> map = new TreeMap<Integer,
String>(Collections.reverseOrder());
        map.putAll(hashMap);
   Set set = map.entrySet();
   Iterator iterator = set.iterator();
   while(iterator.hasNext()) {
        final Map.Entry me = (Map.Entry)iterator.next();
      LinearLayout.LayoutParams params = new
LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT,
LayoutParams.WRAP_CONTENT);
      //create TextViews
      final TextView txtMessage = new TextView(getApplicationContext());
      //create Button
      final Button btnDelete = new Button(getApplicationContext());
      btnDelete.setText(getResources().getString(R.string.btnDelete));
      btnDelete.setContentDescription(getResources().getString(R.string.btnDelete));
      //display subject, body, date, type of message(sent or received)
      String text = "";
      if(records.get(me.getKey()).get(0) == null)
        text = "";
      else
        text = records.get(me.getKey()).get(0) + Html.fromHtml("<br/>");

      text += records.get(me.getKey()).get(1);
      SimpleDateFormat simpleDateFormat = new SimpleDateFormat("d MMMM yyyy'
'HH:MM:ss");
      Date dateTemp = new Date(Long.valueOf(me.getKey().toString()));
      String date = simpleDateFormat.format(dateTemp);
      markMessageRead(me.getKey().toString());
      if(records.get(me.getKey()).get(2) == Integer.toString(INBOX)) {
        text += Html.fromHtml("<br/>") + "Received on " + date;
        txtMessage.setGravity(Gravity.LEFT);
      } else {
        text += Html.fromHtml("<br/>") + "Sent on " + date;
        txtMessage.setGravity(Gravity.RIGHT);
      }
      txtMessage.setWidth(params.width/3);
      txtMessage.setText(text);
      txtMessage.setContentDescription(text);
      txtMessage.setFocusable(true);
      btnDelete.setFocusable(true);
      txtMessage.setLayoutParams(params);
      btnDelete.setPadding(0, 10, 0, 20);
      txtMessage.setLayoutParams(params);

      //to delete the message
      btnDelete.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
                confirmDelete("Are you sure you want to delete this message?", 0,
me.getKey().toString());
        }
      });

      LinearLayout layout = (LinearLayout)findViewById(R.id.textLinearLayout);
      layout.addView(txtMessage);
      layout.addView(btnDelete);
   }
```

```
}
```

Here, address corresponds to the number of the sender/recipient.

8.4 Selecting the recipient for the message

The TextMessagesComposerRecipientApp displays and EditText where the user is expected to enter the number or the name (of the contact) of the recipient. The items in the ListView will be filtered based on the text entered by the user. When a list item is selected, the TextMessagesComposerApp is launched, where the user may enter the body of the message and send it to the selected recipient.

A TextChangedListener is attached to the EditText in order to read out the text entered/deleted and also to filter the items in the ListView.

```
editRecipient.addTextChangedListener(new TextWatcher() {

        @SuppressLint("DefaultLocale")
                    @Override
        public void onTextChanged(CharSequence cs, int arg1, int arg2,
                int arg3) {

                 if(deletedFlag != 1) {
                        if(cs.length() > 0) {
                                //check if keyboard is connected but accessibility services are
disabled
                                if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                                getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS) {
                                        if(cs.toString().substring(cs.length()-1,
cs.length()).matches("(?![@',&])\\p{Punct}")) {
                                                if(editRecipient.getText().toString().matches("-
?\\d+(\\.\\d+)?")) {

       TTS.readNumber(editRecipient.getText().toString());
                                                }
                                                else {
```

```
          TTS.speak(editRecipient.getText().toString());
                                                  }
                                        }
                                        else {
                                                  TTS.speak(cs.toString().substring(cs.length()-1,
cs.length()));
                                        }
                              }
                    }
          }
          else {
                    deletedFlag = 0;
          }

          ArrayList<String> arrayListToBeDisplayed = new ArrayList<String>();
          //check if user entered a letter
          if(!(editRecipient.getText().toString().trim().equals("")) &&
                              !(editRecipient.getText().toString().matches("-?\\d+(\\.\\d+)?"))) {
                    btnProceed.setVisibility(View.GONE);
                    //search if name exists in contacts
                    HashMap<String, ArrayList<String>> listOfNames = new
                                        ContactManager(getApplicationContext()).

          getNamesStartingWith(editRecipient.getText().toString());
                    name = new ArrayList<String>();
                    contactId = new ArrayList<String>();
                    type = new ArrayList<String>();
                    number = new ArrayList<String>();

                    for(int i=0; i<((ArrayList<String>)(listOfNames.get("name"))).size();
i++) {

          if(!contactId.contains(((ArrayList<String>)(listOfNames.get("id"))).get(i).toString()))
                              {

          name.add(((ArrayList<String>)(listOfNames.get("name"))).get(i).toString());
```

```
        contactId.add(((ArrayList<String>)(listOfNames.get("id"))).get(i).toString());

        type.add(((ArrayList<String>)(listOfNames.get("type"))).get(i).toString());

        number.add(((ArrayList<String>)(listOfNames.get("number"))).get(i).toString());
                                    arrayListToBeDisplayed.add(name.get(name.size()-1) +
" " +
                                    type.get(type.size()-1));
                           }
                  }

         }
         else {
                  if(!(editRecipient.getText().toString().trim().equals(""))) {
                           //user entered a number
                           btnProceed.setVisibility(View.VISIBLE);
                           //search if number exists in contacts
                           HashMap<String, ArrayList<String>> listOfNames = new
                                             ContactManager(getApplicationContext()).

        getNamesWithNumber(editRecipient.getText().toString());
                           name = new ArrayList<String>();
                           number = new ArrayList<String>();
                           type = new ArrayList<String>();
                           contactId = new ArrayList<String>();
                           for(int i=0;
i<((ArrayList<String>)(listOfNames.get("name"))).size(); i++) {
                                    if(!number.contains(((ArrayList<String>)

        (listOfNames.get("number"))).get(i).toString()))
                                    {

        name.add(((ArrayList<String>)(listOfNames.get("name"))).
                                                      get(i).toString());

        number.add(((ArrayList<String>)(listOfNames.get("number"))).
                                                      get(i).toString());
        type.add(((ArrayList<String>)(listOfNames.get("type"))).
                                                        get(i).toString());

        contactId.add(((ArrayList<String>)(listOfNames.get("id"))).
                                                        get(i).toString());

        arrayListToBeDisplayed.add(name.get(name.size()-1) + " " +
                                                        type.get(type.size()-1));
                                    }
                            }
                    }
                    else {
                            //empty input
                            btnProceed.setVisibility(View.GONE);
                    }
            }
            adapter = new ContactsAdapter(getApplicationContext(),
arrayListToBeDisplayed);
            namesListView.setAdapter(adapter);
        }

        @Override
        public void beforeTextChanged(CharSequence arg0, int arg1,
              int arg2, int arg3) {

        }

        @Override
        public void afterTextChanged(Editable arg0) {

        }
      });
    }
```

The above code creates an ArrayList consisting of the details of the contact such as name, number, type of number, and contact ID. This ArrayList is passed to the adapter with which the ListView is associated. The user may select a contact from the ListView.
If the number entered by the user is not saved on the device, the Proceed button is displayed which will launch the TextMessagesComposerApp activity.

9.0 Compose

The user is provided with a facility to write the body of the text. The recipient's information is displayed on the screen. When the **Send** button is clicked, the message is sent to the recipient and the user is informed about the status.

```
try {
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(), 0,
                                new Intent(SENT), 0);
        PendingIntent deliveredPI = PendingIntent.getBroadcast(getApplicationContext(),
                                0, new Intent(DELIVERED), 0);
        //---when the SMS has been sent---
        registerReceiver(new BroadcastReceiver(){
                @Override
                public void onReceive(Context arg0, Intent arg1) {
switch (getResultCode()) {
                                case Activity.RESULT_OK:
                        //check if keyboard is connected but accessibility services are
disabled
                        if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                        getResources().getConfiguration().keyboard !=
                                                Configuration.KEYBOARD_NOKEYS)
Utils.giveFeedback(getApplicationContext(),
                                        getResources().getString(R.string.sentSms));
                                        Toast.makeText(getApplicationContext(),
                                        getResources().getString(R.string.sentSms),
                                        Toast.LENGTH_SHORT).show();
                                        break;
                                case SmsManager.RESULT_ERROR_NO_SERVICE:
                        //check if keyboard is connected but accessibility services are
disabled
```

```
                              if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                        getResources().getConfiguration().keyboard !=
                                                  Configuration.KEYBOARD_NOKEYS)
                                        Utils.giveFeedback(getApplicationContext(),
                                        getResources().getString(R.string.noService));
                                        Toast.makeText(getApplicationContext(),
                                        getResources().getString(R.string.noService),
                                        Toast.LENGTH_SHORT).show();
                                        break;
                                   case SmsManager.RESULT_ERROR_RADIO_OFF:
                              //check if keyboard is connected but accessibility services are
disabled
                              if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                        getResources().getConfiguration().keyboard !=
                                        Configuration.KEYBOARD_NOKEYS)
                                        Utils.giveFeedback(getApplicationContext(),
                                        getResources().getString(R.string.radioOff));
                                        Toast.makeText(getApplicationContext(),
                                        getResources().getString(R.string.radioOff),
                                        Toast.LENGTH_SHORT).show();
                                        break;
                                        default:
                              //check if keyboard is connected but accessibility services are
disabled
                              if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                        getResources().getConfiguration().keyboard !=
                                        Configuration.KEYBOARD_NOKEYS)
                                        Utils.giveFeedback(getApplicationContext(),
                                        getResources().getString(R.string.smsNotSent));
                                        Toast.makeText(getApplicationContext(),
                                        getResources().getString(R.string.smsNotSent),
                                        Toast.LENGTH_SHORT).show();
                                        break;
                              }
                    }
          }, new IntentFilter(SENT));
          //---when the SMS has been delivered---
          registerReceiver(statusReceiver, new IntentFilter(DELIVERED));
```

```
            SmsManager sms = SmsManager.getDefault();
            sms.sendTextMessage(TextMessagesComposerApp.this.number, null,
                                                   editMessage.getText().toString(), sentPI, deliveredPI);
    }
    catch(Exception e) {
             e.printStackTrace();
    }
}
```

statusReceiver is a BroadcastReceiver that listens to the event when the SMS is delivered to the recipient. It announces the status if the keyboard is connected to the device. A Toast notification is also displayed informing the user about the status.

```
statusReceiver = new BroadcastReceiver(){
@Override
            public void onReceive(Context arg0, Intent arg1) {
                        switch (getResultCode()) {
                                    case Activity.RESULT_OK:
                                    //check if keyboard is connected but accessibility services are
disabled
                                    if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                                getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                                Utils.giveFeedback(getApplicationContext(),

getResources().getString(R.string.smsDelivered));
                                                Toast.makeText(getApplicationContext(),
                                                getResources().getString(R.string.smsDelivered),
                                                Toast.LENGTH_SHORT).show();
                                    break;
                                    case Activity.RESULT_CANCELED:
                                    //check if keyboard is connected but accessibility services are
disabled
                                    if(!Utils.isAccessibilityEnabled(getApplicationContext()) &&
                                                getResources().getConfiguration().keyboard !=
Configuration.KEYBOARD_NOKEYS)
                                                Utils.giveFeedback(getApplicationContext(),
                                                    getResources().getString(R.string.smsNotDelivered));
                                                    Toast.makeText(getApplicationContext(),
getResources().getString(R.string.smsNotDelivered),
Toast.LENGTH_SHORT).show();
                                break;
                                }
                    }
          };
```

10.0 Call Log

The Call Log app is lists the outgoing, incoming and missed calls. The total duration of incoming and outgoing calls are displayed as well. When the user clicks on a log, the user is provided with options to call the number or contact, send a text message, delete the log from records, view call history, and to view the details of the contact or to save the number to contacts.

When the activity is loaded, all the call logs consisting of the dialed numbers or contacts are displayed. When **In** is selected, the ListView displays the numbers or contacts from which the calls were received along with the time of the call. When **Missed** is selected, the ListView displays the numbers or contacts from which the calls were not answered along with the time of the call. When **All** is selected, the ListView displays all the logs, outgoing as well as incoming. The total duration of incoming, and outgoing calls is displayed when **Total Duration** button is clicked.

getCallLogs() accepts the type of the log as a parameter and loads the results in the ListView accordingly. The total incoming/outgoing call duration is calculated by retrieving all the incoming/outgoing calls and adding the duration of each call.

```
void getCallLogs(int type) {
                    Cursor cursor = null;
                    Message message = new Message();
                    Bundle bundle = new Bundle();
                    bundle.putInt("type",type);
                    switch(type) {
                    case TYPE_DIALED:
                              cursor =
getContentResolver().query(Uri.parse("content://call_log/calls"), null,
                                        "type = " + CallLog.Calls.OUTGOING_TYPE, null, "date
DESC");
                        break;
                case TYPE_RECEIVED:
                        cursor =
getContentResolver().query(Uri.parse("content://call_log/calls"), null,
                                        "type = " + CallLog.Calls.INCOMING_TYPE, null, "date
DESC");
                        break;
                case TYPE_MISSED:
                        cursor =
getContentResolver().query(Uri.parse("content://call_log/calls"), null,
                                        "type = " + CallLog.Calls.MISSED_TYPE, null, "date
DESC");
                        break;
                case TYPE_ALL:
                        cursor =
getContentResolver().query(Uri.parse("content://call_log/calls"), null,
                                        null, null, "date DESC");
                        break;
                case TYPE_TIME:
                        cursor =
getContentResolver().query(Uri.parse("content://call_log/calls"),
                                        new String[] {CallLog.Calls.DURATION}, "type = " +
CallLog.Calls.OUTGOING_TYPE,
                                        null, null);
                        long totalOutgoingTime = 0, totalIncomingTime = 0;
                        while(cursor.moveToNext()) {
                                totalOutgoingTime +=
cursor.getLong(cursor.getColumnIndex(CallLog.Calls.DURATION));
                        }
                        cursor =
getContentResolver().query(Uri.parse("content://call_log/calls"), new String[]
                                        {CallLog.Calls.DURATION}, "type = " +
CallLog.Calls.INCOMING_TYPE, null, null);
                        while(cursor.moveToNext()) {
                                totalIncomingTime +=
cursor.getLong(cursor.getColumnIndex(CallLog.Calls.DURATION));
```

```
                    }
                    bundle.putLong("outtime", totalOutgoingTime);
                    bundle.putLong("intime", totalIncomingTime);
                    message.setData(bundle);
                    cursor.close();
                    cursor = null;
                    break;
            }
            int i = 0;
            if(cursor != null && cursor.getCount() == 0) {
                    message = new Message();
                    message.setData(null);
            }
            else if(cursor != null){
                    while(cursor.moveToNext()) {
                            Bundle values = new Bundle();

    if(cursor.getString(cursor.getColumnIndex(CallLog.Calls.CACHED_NAME)) != null)
                                    values.putString("name",
cursor.getString(cursor.getColumnIndex(CallLog.Calls.
                                            CACHED_NAME)) + " ");
                            else
                                    values.putString("name", "");

    if(cursor.getString(cursor.getColumnIndex(CallLog.Calls.CACHED_NUMBER_LAB
EL)) != null)

    values.putString("label",cursor.getString(cursor.getColumnIndex(CallLog.Calls.
                                            CACHED_NUMBER_LABEL)) +
Html.fromHtml("<br/>"));
                            else
                                    values.putString("label",
Html.fromHtml("<br/>").toString());

    if(cursor.getString(cursor.getColumnIndex(CallLog.Calls.NUMBER)) != null)
                                    values.putString("number", cursor.getString(cursor.
                                            getColumnIndex(CallLog.Calls.NUMBER))
+ " ");
```

```
                                else
                                        values.putString("number", "");

        if(cursor.getString(cursor.getColumnIndex(CallLog.Calls.DATE)) != null) {
                                        Date date = new
Date(Long.valueOf(cursor.getString(cursor.
                                                        getColumnIndex(CallLog.Calls.DATE))));
                                        SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("d MMMM yyyy' 'HH:MM:ss");
                                        values.putString("date", Html.fromHtml("<br/>") +
                                                        simpleDateFormat.format(date) + " ");
                                }
                                else
                                        values.putString("date", "");

        if(Integer.toString(cursor.getInt(cursor.getColumnIndex(CallLog.Calls._ID))) != null)
                                        values.putString("id",
cursor.getString(cursor.getColumnIndex(CallLog.Calls._ID)));
                                else
                                        values.putString("id", "");
                                bundle.putBundle(Integer.toString(i), values);
                                i++;
                        }
                        message.setData(bundle);
                }
                handler.sendMessage(message);
        }
```

When the user clicks on an item in the ListView, the CallLogOptions activity is launched. startNewActivity() is called when an onClick event is triggered on one of the following buttons. The view is passed as a parameter. Based on the id of the view, the corresponding action is performed.

```
void startNewActivity(View view) {
        Intent intent;
        switch (view.getId()) {
        case R.id.btnCall:
                // pass number to dialer app
                intent = new Intent(getApplicationContext(), PhoneDialerApp.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                intent.putExtra("call", CallLogOptions.this.number);
                startActivity(intent);
                finish();
                break;
        case R.id.btnSendSMS:
                intent = new Intent(getApplicationContext(),
                                TextMessagesComposerApp.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                intent.putExtra("number", CallLogOptions.this.number);
                /* pass the name and type of number if the number is stored in contacts */
                startActivity(intent);
                break;
        case R.id.btnViewCallHistory:
                intent = new Intent(getApplicationContext(), CallLogHistory.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                intent.putExtra("number", CallLogOptions.this.number);
                // pass the name if the number is stored in contacts
                intent.putExtra("id", CallLogOptions.this.id);
                startActivity(intent);
                break;
        case R.id.btnDeleteFromLog:
                if (getContentResolver().delete(Uri.parse("content://call_log/calls"),
                                CallLog.Calls._ID + "=?",
                                new String[] { CallLogOptions.this.id }) != 0) {
                        // display or read out the appropriate message and destroy the
                        // activity
                        finish();
                } else {
                        // display or read out the approprite message if the log could not
                        // be deleted.
                }
                break;
        case R.id.btnContact:
                if (!CallLogOptions.this.name.equals("")) {
                        // view contact
                        intent = new Intent(getApplicationContext(),
                                                ContactsDetailsMenu.class);
                        /* pass the name and number of the contact and launch the activity */
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        startActivity(intent);
                } else {
                        // save to contacts
                        intent = new Intent(getApplicationContext(), SaveContact.class);
                        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                        intent.putExtra("number", CallLogOptions.this.number);
                        startActivity(intent);
                }
                finish();
                break;
        }
}
```

10.1 Call Log History

The Call Log History will display the previous call logs from a particular number. getCallLogHistory() takes as parameter, the number whose logs should be displayed. The latest call is displayed at the top of the list.

```
void getCallHistory(String number) {
                Cursor cursor = getContentResolver().query(
                                Uri.parse("content://call_log/calls"), null,
                                CallLog.Calls.NUMBER + " = ?", new String[] { number },
                                "date DESC");
                int i = 0;
                Message message = new Message();
                Bundle bundle = new Bundle();
                while (cursor.moveToNext()) {
                        Bundle values = new Bundle();
                        if (cursor.getString(cursor.getColumnIndex(CallLog.Calls.DATE)) !=
null) {
                                Date date = new Date(Long.valueOf(cursor.getString(cursor
                                                .getColumnIndex(CallLog.Calls.DATE))));
```

```
                                        SimpleDateFormat simpleDateFormat = new
SimpleDateFormat(
                                                            "d MMMM yyyy' 'HH:MM:ss");
                                        values.putString("date", Html.fromHtml("<br/>")
                                                            + simpleDateFormat.format(date) + " ");
                              } else
                                        values.putString("date", "");
                              if (cursor.getString(cursor

          .getColumnIndex(CallLog.Calls.CACHED_NUMBER_LABEL)) != null) {
                                        String type = cursor.getString(cursor

          .getColumnIndex(CallLog.Calls.CACHED_NUMBER_LABEL));
                                        values.putString("type", type);
                              } else
                                        values.putString("type", "");
                              values.putInt("status",

          cursor.getInt(cursor.getColumnIndex(CallLog.Calls.TYPE)));
                              bundle.putBundle(Integer.toString(i), values);
                              i++;
                    }
                    message.setData(bundle);
                    handler.sendMessage(message);
          }
```