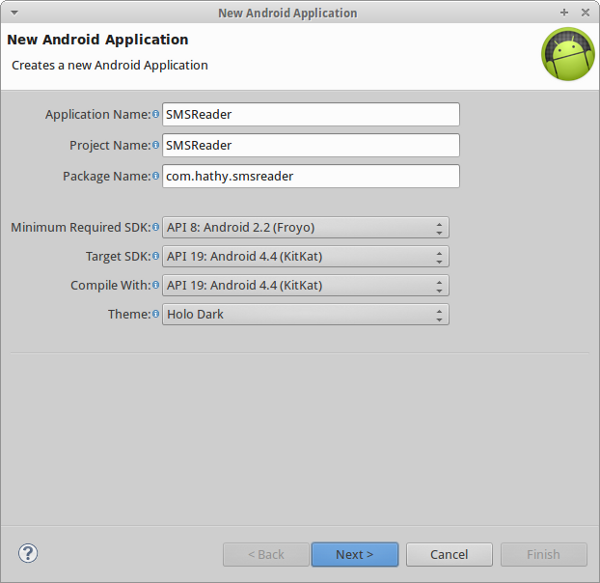
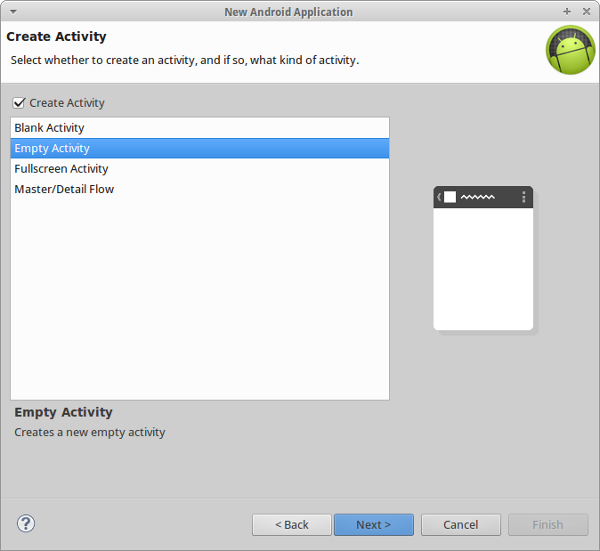
Use Text-to-Speech on Android to Read Out Incoming Messages

In this tutorial, you'll learn how to create a simple app—with a minimalist user interface—that can receive text messages and read them to the user.

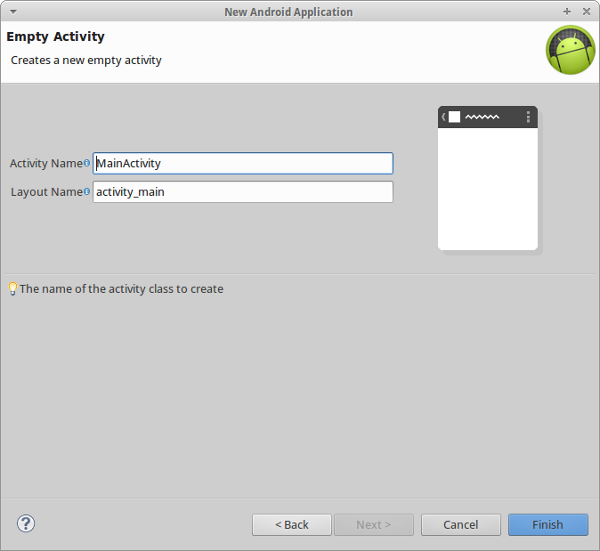
1. Create a New Project



This app will have one Activity. Select Create Activity and choose Empty Activity.



Name it MainActivity and click Finish.



2. Edit the Manifest

<uses-permission android:name="android.permission.READ\_SMS"/>

<uses-permission android:name="android.permission.RECEIVE\_SMS"/>

<uses-permission android:name="android.permission.READ\_CONTACTS"/>

This app is going to have only one screen orientation, portrait. Therefore, edit the activity tag and add the following attribute to it:

android:screenOrientation="portrait"

The manifest is now complete.

3. Edit strings.xml

It is not absolutely necessary, but storing all strings that the application uses in the res/values/strings.xml file is a good practice. Edit this file so that it has the following contents:

<?xml version="1.0" encoding="utf-8"?>

<resources>

    <string name="app\_name">SMSReader</string>

    <string name="sms\_label">Latest SMS</string>

    <string name="none">None</string>

    <string name="speech\_toggle\_on">START SPEAKING</string>

    <string name="speech\_toggle\_off">STOP SPEAKING</string>

    <string name="start\_speaking">Okay! I will read your messages out loud for you now.</string>

    <string name="stop\_speaking">Okay! I will stay silent now.</string>

</resources>

Most of these strings are used in the next step.

4. Edit the Layout

Edit res/layout/activity\_main.xml to add the following:

* a TextView to display the name of the person who sent the latest SMS
* a TextView to display the contents of the latest SMS
* a ToggleButton to toggle speech output on and off

After adding code for positioning and styling these elements, your file should have the following contents:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      xmlns:tools="<http://schemas.android.com/tools>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      tools:context="${packageName}.${activityClass}"      android:background="#99CC00"      >        <TextView          android:id="@+id/sms\_sender"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_alignParentTop="true"          android:layout\_centerHorizontal="true"          android:layout\_marginTop="20dp"          android:text="@string/sms\_label"          android:textColor="#ffffff"          android:textAppearance="?android:attr/textAppearanceSmall" />        <TextView          android:id="@+id/sms\_text"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_below="@+id/sms\_sender"          android:layout\_centerHorizontal="true"          android:layout\_marginTop="10dp"          android:text="@string/none"          android:textAppearance="?android:attr/textAppearanceLarge"          android:textColor="#ffffff"          />        <ToggleButton          android:id="@+id/speechToggle"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:layout\_centerHorizontal="true"          android:layout\_centerVertical="true"          android:textOff="@string/speech\_toggle\_on"          android:textOn="@string/speech\_toggle\_off"          />    </RelativeLayout> |

The layout for our application is now complete.

5. Create a Helper Class

We're now going to create a helper class for the TTS engine. Create a new Java class and call it Speaker.java. This class is used to avoid calling the TTS API directly from the Activity.

This class implements the OnInitListener interface so that it knows when the TTS engine is ready. We store this ready state in a boolean variable named ready. We use another boolean variable named allowed whose value is true only if the user has allowed the TTS engine to speak. We also add methods to get and set the value of this variable. At this point, Speaker.java should have the following contents:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20 | public class Speaker implements OnInitListener {        private TextToSpeech tts;      private boolean ready = false;      private boolean allowed = false;        public Speaker(Context context){          tts = new TextToSpeech(context, this);      }        public boolean isAllowed(){          return allowed;      }        public void allow(boolean allowed){          this.allowed = allowed;      }  } |

The OnInitListener interface has only one method, onInit. This method is called when the TTS engine has been initialized. The status parameter lets us know if the initialization was successful. Once we know that the initialization was successful, we set the language of the TTS engine. This is important to produce speech that is comprehensible. Add the following code:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | @Override  public void onInit(int status) {      if(status == TextToSpeech.SUCCESS){            tts.setLanguage(Locale.US);          ready = true;      }else{          ready = false;      }  } |

Next, we add a method named speak, which uses the engine to read out any text that is passed to it. Before doing so, it checks if both the allowed and the ready values are true. The speech it generates is placed in the notification stream.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12 | public void speak(String text){        // Speak only if the TTS is ready and the user has allowed speech        if(ready && allowed) {          HashMap<String, String> hash = new HashMap<String,String>();          hash.put(TextToSpeech.Engine.KEY\_PARAM\_STREAM,                  String.valueOf(AudioManager.STREAM\_NOTIFICATION));          tts.speak(text, TextToSpeech.QUEUE\_ADD, hash);      }  } |

We then add a method that plays silence for a specified duration. Using this method, we can add pauses to the speech to make it sound a little clearer. Add the following code to the implementation:

public void pause(int duration){

    tts.playSilence(duration, TextToSpeech.QUEUE\_ADD, null);

}

Finally, add a method to free up resources when the TTS engine is no longer needed.

// Free up resources

public void destroy(){

    tts.shutdown();

}

6. Edit the Activity Class

Edit MainActivity.java and declare all the views that we mentioned in the layout. Declare two integers, LONG\_DURATION and SHORT\_DURATION. These are merely values that are passed to the Speaker's pause method.

Also declare a CHECK\_CODE integer. It's value isn't important. It is passed to the startActivityforResult method and then used to identify the result.

Lastly, declare a Speaker object and a BroadcastReceiver object.

At this point, your class should look like this:

**public class MainActivity extends Activity {**

    private final int CHECK\_CODE = 0x1;

    private final int LONG\_DURATION = 5000;

    private final int SHORT\_DURATION = 1200;

    private Speaker speaker;

    private ToggleButton toggle;

    private OnCheckedChangeListener toggleListener;

    private TextView smsText;

    private TextView smsSender;

    private BroadcastReceiver smsReceiver;

}

Add a method to check if a TTS engine is installed on the device. The check is performed by making use of the result of another Activity.

**private void checkTTS(){**

    Intent check = new Intent();

    check.setAction(TextToSpeech.Engine.ACTION\_CHECK\_TTS\_DATA);

    startActivityForResult(check, CHECK\_CODE);

}

When the result of startActivityForResult arrives, the onActivityResult method is called. Therefore, we need to override it. In this method, if the result is positive, we initialize the Speaker object. If no TTS engine is installed, we redirect the user to install it.

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

    if(requestCode == CHECK\_CODE){

        if(resultCode == TextToSpeech.Engine.CHECK\_VOICE\_DATA\_PASS){

            speaker = new Speaker(this);

        }else {

            Intent install = new Intent();

            install.setAction(TextToSpeech.Engine.ACTION\_INSTALL\_TTS\_DATA);

            startActivity(install);

        }

    }

}

Whenever there are new messages, its onReceive method is called. We parse the messages, which arrive as byte arrays, using the SmsMessage class. Once the message is parsed, we use methods like getDisplayMessageBody and getOriginatingAddress to extract meaningful information out of it.

With this information, we generate the text that the TTS engine should read out. We pause for LONG\_DURATION before reading out a new SMS and for SHORT\_DURATION between the utterances of the SMS sender's name and the SMS's body.

private void initializeSMSReceiver(){

    smsReceiver = new BroadcastReceiver(){

        @Override

        public void onReceive(Context context, Intent intent) {

            Bundle bundle = intent.getExtras();

            if(bundle!=null){

                Object[] pdus = (Object[])bundle.get("pdus");

                for(int i=0;i<pdus.length;i++){

                    byte[] pdu = (byte[])pdus[i];

                    SmsMessage message = SmsMessage.createFromPdu(pdu);

**String text = message.getDisplayMessageBody();**

**String sender = getContactName(message.getOriginatingAddress());**

                    speaker.pause(LONG\_DURATION);

                    speaker.speak("You have a new message from" + sender + "!");

                    speaker.pause(SHORT\_DURATION);

                    speaker.speak(text);

                    smsSender.setText("Message from " + sender);

                    smsText.setText(text);

                }

            }

        }

    };

}

We can only extract the phone number of the sender from the message. To map this number to a contact's name, we have to make use of the user's contacts. The following method queries the contacts data. If the phone number is not available in the user's contacts, then it simply returns the string unknown number:

private String getContactName(String phone){

    Uri uri = Uri.withAppendedPath(PhoneLookup.CONTENT\_FILTER\_URI, Uri.encode(phone));

    String projection[] = new String[]{ContactsContract.Data.DISPLAY\_NAME};

    Cursor cursor = getContentResolver().query(uri, projection, null, null, null);

    if(cursor.moveToFirst()){

        return cursor.getString(0);

    }else {

        return "unknown number";

    }

}

Before the BroadcastReceiver can be used, it has to be registered. In the following method, we create an IntentFilter for incoming text messages and then register our smsReceiver for it:

private void registerSMSReceiver() {

**IntentFilter intentFilter = new IntentFilter("android.provider.Telephony.SMS\_RECEIVED");**

**registerReceiver(smsReceiver, intentFilter);**

}

Next, we create the onCreate method. Here's where we initialize all the objects we declared. We initialize the toggleListener to set the value of allowed in the Speaker class.

After these initializations, we call the checkTTS, initializeSMSReceiver, and registerSMSReceiver methods.

@Override

protected void **onCreate(**Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity\_main);

    toggle = (ToggleButton)findViewById(R.id.speechToggle);

    smsText = (TextView)findViewById(R.id.sms\_text);

    smsSender = (TextView)findViewById(R.id.sms\_sender);

    toggleListener = new OnCheckedChangeListener() {

        @Override

        public void onCheckedChanged(CompoundButton view, boolean isChecked) {

            if(isChecked){

                speaker.allow(true);

                speaker.speak(getString(R.string.start\_speaking));

            }else{

                speaker.speak(getString(R.string.stop\_speaking));

                speaker.allow(false);

            }

        }

    };

    toggle.setOnCheckedChangeListener(toggleListener);

    checkTTS();

**initializeSMSReceiver();**

**registerSMSReceiver();**

}

Finally, in the onDestroy method of the activity, we unregister our receiver and shut down the TTS engine to free up resources.

@Override

protected void onDestroy() {

    super.onDestroy();

    unregisterReceiver(smsReceiver);

    speaker.destroy();

}

7. Run and Test

The app is now ready to be tested. Compile and run it on a physical Android device. Tap the toggle button to enable voice and send yourself an SMS from another phone or ask one of your friends to do so. You should soon be able to hear your phone read out the SMS for you.

Conclusion

In this tutorial, you have learnt to not only use the text-to-speech API, but also to use broadcast receivers and to make sense out of raw SMS data. You can now go on