# 1. Scope

The tutorial will cover implementation of a text to speech application using android Text-To-Speech API. This tutorial targets Android 2.1 (Eclair) handsets. Hence, uses Android 2.1-update1 (API Level 7). At the end of this tutorial you will become familiar with the Android Text-To-Speech API, layouts, some of the essential UI components and event handling.

## 1.1. Overview of the Application

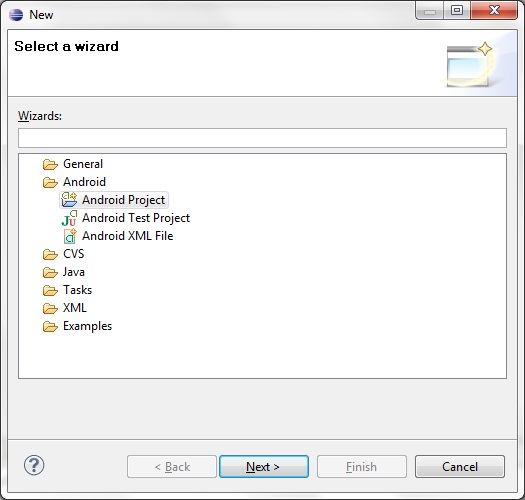
It will be a utility that will speak the text provided on a text area during a button press. The application will contain a text area and a button and its background will be replaced by an image as opposed to using the default black background. It will also hold menu with two menu items, one for closing the application and the second menu item to display an about dialog.

# 2. Pre-requisites

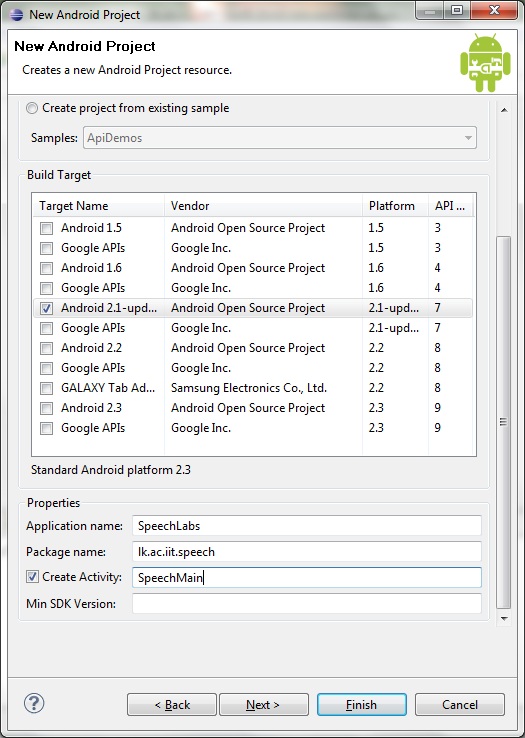
* Java JDK 6 (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
* Eclipse (v3.5 Galileo is preferred) with Android Development Tools (ADT) Plug-in for eclipse (Eclipse Update Site: https://dl-ssl.google.com/android/eclipse/)
* Android SDK (http://developer.android.com/sdk/index.html) and AVD (Android Virtual Device)
* This tutorial will use Android 2.1-update1 (API Level 7)

# 3. Create the Project

**File -> New -> Other -> Android Project**

****

Click **Next** on the above dialog and continue to fill out project details. Provide ‘***SpeechLabs***’ as the Project name and Application name, ‘***lk.ac.iit.speech***’ as the package name, ‘***SpeechMain***’ for the Activity name and select **Android 2.1-update1** as the **build target**. Rest of the configurations can be left as default and click **Finish** to create the project.

****

# 4. Copy the Resource Files

The application is to use a custom icon (icon.png 72x72px) and a background image (app\_bg.jpg 660x854px). These files need to be copied to **drawable** folder under res folder. **If** this folder is **not found**, **right-click** on **res** folder, select **new->folder** and name it ‘***drawable*’**.

# resources.jpgcopy_resources.jpg

# 5. Setup the UI Layout

Now the background of the Activity (‘***SpeechMain***’) and the layout of it can be setup by introducing the new UI components (Text View, Edit Text and a Button). Hence, remove the default Text View tag provided on **main.xml** found under **layout** folder and introduce the new elements as found below. It is also noticeable that ‘***android:backround***’ property has introduced to **LinearLayout tag**. Figure 4 shows the Graphical Layout Perspective of main.xml after the modifications.

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

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:orientation=*"vertical"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*

android:background=*"@drawable/app\_bg"*

>

<TextView android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"* android:text=*"Text to Speak"* />

<EditText android:id=*"@+id/txtSpeak"* android:layout\_width=*"fill\_parent"*

android:layout\_height=*"150px"* android:gravity=*"top"* />

<Button android:layout\_marginTop=*"30px"*

android:layout\_marginLeft=*"28px"*

android:id=*"@+id/btnSpeak"*

android:layout\_width=*"200px"* android:layout\_height=*"45px"*

android:text=*"Speak"* />

</LinearLayout>

**Tip 1:** Android Market connects its users and developers. Users get to download applications over-the-air (OTA) without using a PC and it also allows developers to publish and distribute their android applications for Android Ecosystem. Alternative app markets include AppBrain, AndroLib, Handango and AndSpot.

**Note:** The ids specified above (i.e. **txtSpeak** and **btnSpeak**) should be available on R.java once the project is refreshed or cleaned. The file can located in **lk.ac.iit.speech** package of **gen** folder.

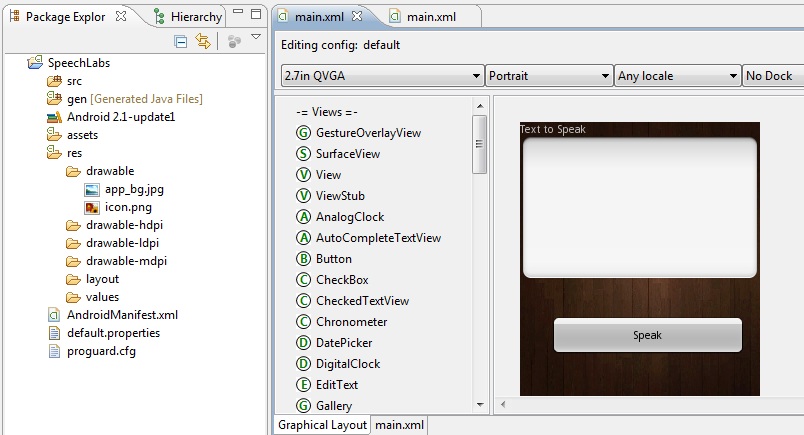


Figure 4: Graphical Layout Perspective of modified main.xml

# 6. Introduce the Menu

The menu items will be residing in a separate xml file and it is a best practice to have the menus in a separate folder. Hence, a folder named menu will be used for creating the this xml. To create this folder, **right click** on **res** folder -> **new -> folder** and name it ‘***menu***’.

Once menu folder is created, the xml to hold the menu items also needs to be created. To create this file, **right click** on **menu** select **new -> file** and name it ‘**menu.xml**’. Once this file is created, **select** its **Layout Perspective** and **click** on **add** button. In the resulting dialog **select Item** and **Click OK.**

Once the new item is added, provide a name for it (‘***About***’). Then follow the same steps to create the second menu item (‘***Close***’). **Note:** When the second menu item is created make sure *Create a new element at the top level, In Menu* is checked. The figure below shows the completed menu.xml in Layout perspective.

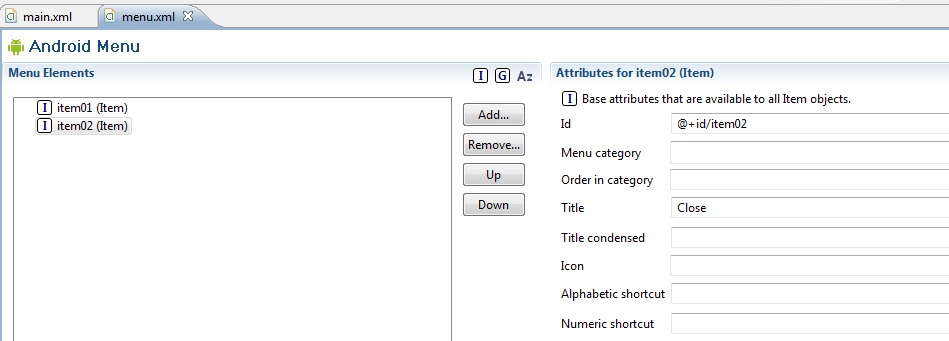


Figure 5: Completed Menu.xml in its Layout perspective

**Tip 2:** The **drawable** (in other cases under **drawable-hdpi** /**drawable-ldpi** /**drawable-mdpi**) folder residing inside **res** folder holds all the resource files used by the application such as image, sound and other media files. **Names** of these resource files should be **unique** and their **file names are not allowed to start with a number**. Inside the **gen** folder, the package architecture used by the main activity of the application is replicated and a Resource Java (**R.java**) is **generated**. This class contains **IDs** **for each UI element** used by the application and it also provides a **mapping of resource name to a constant integer** (i.e. static final integer) which can be used inside the code to call or make use of the resource files.

# 7. Handling the button press and utilizing the Text To Speech API

At this stage the layout and all UI elements of the application has been completed and we can move to its code. Initially creating an instance of the text To Speech Class and handling the button press will be implemented. Following this, handling menu items via code will be discussed. The code below shows **initiation of the button and text field instances** (using the id provided on R.java) and **onClick()** event handler for **speak button**. Now an instance of the Text-To-Speech class be created and called when the speak button is pressed to **speak out** the text found in **textSpeak** **field**. However in order to initiate the text to speech it is essential to implement OnInitListener interface and override its **onInit()** method. The code on next page shows its implementation.

**package** lk.ac.iit.speech;

**import** java.util.Locale;

**import** lk.ac.iit.speech.R;

**import** android.app.Activity;

**import** android.os.Bundle;

**import** android.speech.tts.TextToSpeech;

**import** android.speech.tts.TextToSpeech.OnInitListener;

**import** android.view.View;

**import** android.widget.Button;

**import** android.widget.EditText;

**public** **class** SpeechMain **extends** Activity **implements** OnInitListener {

EditText textSpeak;

Button buttonSpeak;

**private** **static** TextToSpeech *tts*;

/\*\* Called when the activity is first created. \*/

@Override

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

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

textSpeak = (EditText) findViewById(R.id.*txtSpeak*);

buttonSpeak = (Button) findViewById(R.id.*btnSpeak*);

*tts* = **new** TextToSpeech(**this**, **this**);

buttonSpeak.setOnClickListener(**new** View.OnClickListener() {

**public** **void** onClick(View view) {

**if** (!(textSpeak.getText().toString().equals(""))) {

// speak out the text found on textSpeak

*tts*.speak(textSpeak.getText().toString(),

TextToSpeech.*QUEUE\_FLUSH*, **null**);

} **else** {

// ask to insert some text

*tts*.speak("Please enter soem ext to speak",

TextToSpeech.*QUEUE\_FLUSH*, **null**);

}

}

});

}

**public** **void** onInit(**int** status) {

*tts*.setLanguage(Locale.*UK*);

*tts*.speak("Welcome to SpeechLabs", TextToSpeech.*QUEUE\_FLUSH*, **null**);

}

# 8. Handling the Menu Items

Now the menu items can be handled via code. The methods below are introduced to the above activity for the purpose.

**public** **boolean** onCreateOptionsMenu(Menu menu) {

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.*menu*, menu);

**return** **true**;

}

**public** **boolean** onOptionsItemSelected(MenuItem item) {

**switch** (item.getItemId()) {

**case** R.id.*item01*:

finish();

**break**;

**case** R.id.*item02*:

AlertDialog.Builder alert = **new** AlertDialog.Builder(**this**);

alert.setCancelable(**true**);

alert.setTitle("About Speech Labs");

alert.setMessage("This is a text to speech utility developed at IIT.");

alert.setPositiveButton("OK",

**new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog,

**int** whichButton) {

}

});

alert.show();

**break**;

}

**return** **true**;

}

# 09. A screen shot of the application

# screenshot.jpg

# 10. Going Further

If time permits signing this application using a developer keystore and preparing it for publishing on Android Market will be discussed.