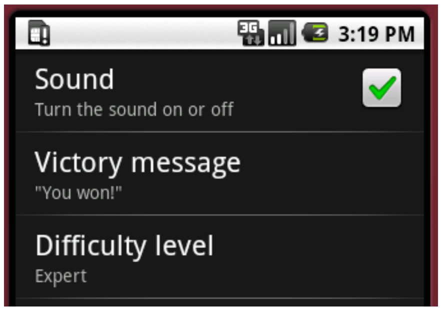
Android Application Programming

Tutorial 6: App Settings

Introduction



Sound
Turn the sound on or off ✓

Victory message
"You won!"

Difficulty level
Expert

Most Android applications have a Settings menu option which allows users to set the application's settings/preferences. This is so common that a specialized Activity class was developed just for changing application settings (`android.preference.PreferenceActivity`). In this tutorial you will learn how to create applications settings using the `PreferenceActivity`, and you will learn how to start another activity using an Intent.

Setup Your Project

Download and import the Andriod project into Eclipse which implements the extra challenge from the previous tutorial:

1. Download the zip file **AndroidTicTacToeSolution4.zip** from Google Code University.
2. Unzip **AndroidTicTacToeSolution4.zip** in a directory of your choosing.
3. In Eclipse, choose **File → Import… → Existing Projects into Workspace…**
4. Select "Select root directory:" and click **Browse**.
5. Go to the **AndroidTicTacToeSolution4** directory that was created in step 2 and click OK.
6. Make sure "Copy projects into workspace" is checked, then click **Finish**.

Create the Preference XML and PreferenceActivity

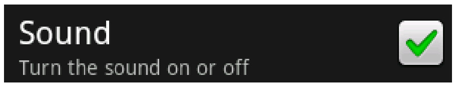
The PreferenceActivity simplifies the creation of a settings screen for your application. It uses an XML file (preferences.xml) to list the various application settings. The preferences are saved using the `SharedPreferences` class, just as they were in the previous tutorial. In some cases, very little to no code needs to be written to change application settings.

The first thing you need to do is create an XML file that lists the three application settings for our tic-tac-toe game:

1. Create a `res/xml` directory.

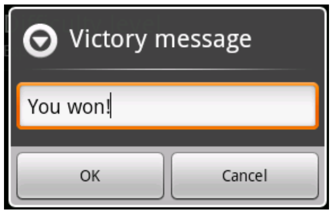
2. Create a `res/xml/preferences.xml` file. We'll be creating three preferences:

   1) a `CheckBoxPreference` for turning the sound on and off,

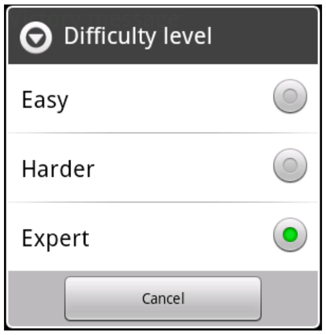


Sound
Turn the sound on or off ✓

2) an `EditTextPreference` for setting the text that is displayed when the user wins, and



3) a `ListPreference` for setting the AI's difficulty level.



These three are not the only types of preferences, but they are probably the most popular types of preferences.

3. Type the XML below into `preferences.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

   <CheckBoxPreference
         android:key="sound"
         android:title="Sound"
         android:defaultValue="true"
         android:summary="Turn the sound on or off" />

   <EditTextPreference
         android:key="victory_message"
         android:summary=""
         android:defaultValue="@string/result_human_wins"
         android:title="Victory message"  />

    <ListPreference
         android:key="difficulty_level"
         android:title="Difficulty level"
         android:summary=""
         android:defaultValue="@string/difficulty_expert"
         android:entries="@array/list_difficulty_level"
         android:entryValues="@array/list_difficulty_level" />

</PreferenceScreen>
```

The `PreferenceScreen` can have any number of preferences inside it. If there are a large number of preferences which can be organized by type, it's a good idea to organize these into `PreferenceCategory`'s. Since we only have three preferences, a `PreferenceCategory` is not necessary.

Note that some of the default values for the preferences are coming strings that we defined earlier in the `strings.xml` file (`result_human_wins` and `difficulty_expert`). The ListPreference will get its values from an arrays.xml file we'll create in the next step.

4. Create a `res/values/arrays.xml` file with the following XML:

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

    <string-array name="list_difficulty_level">
        <item>@string/difficulty_easy</item>
        <item>@string/difficulty_harder</item>
        <item>@string/difficulty_expert</item>
    </string-array>

</resources>
```

This file is creating a list that will be displayed when selecting the difficulty level, and it uses strings from the `strings.xml` file rather than hard-coding text that may need to be changed sometime in the future.

5. Now add a Java class to your project called `Settings` which extends the `PreferenceActivity` class. This class should specify in the `onCreate()` method that the `preferences.xml` file will be used to display the list of preferences:

```java
public class Settings extends PreferenceActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.preferences);
   }
}
```

Now we need to display the settings which we'll do in the next step.

Display the Settings

Since we are using the Preferences to set the difficulty level, we don't need to do this using the application menu anymore. We'll replace the Difficulty menu button with one that displays the settings.

1. In `res/menu/options_menu.xml`, replace the Difficulty menu with a Settings menu option. You will also need to add an image to `res/drawable` for representing the settings option.

```xml
<item android:id="@+id/settings"
      android:title="Settings"
      android:icon="@drawable/menu_settings" />
```

2. In `AndroidTicTacToe.java`, remove the `final int DIALOG_DIFFICULTY_ID` and the code for creating the difficulty level dialog box in `onCreateDialog()`.

3. Replace the code for handing the Difficulty menu option in `onOptionsItemSelected()` with code that will launch the Settings activity. It will use an Intent to start the activity:

```java
case R.id.settings:
    startActivityForResult(new Intent(this, Settings.class), 0);
    return true;
```

The startActivityForResult() function indicates that we would like to be notified when the Settings activity is exited (when the user presses the Android device's Back button). This is important since we'll need to change some class-level variables if the sound is turned on or off or if the difficulty level is changed. There's also a startActivity() function which could be used if we don't need to be notified when the activity is closed.

4. Now create the onActivityResult() method which will be called when the Settings activity is exited. The resultCode will be set to RESULT_CANCELED if the user clicks the Back button on their device.

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

   if (requestCode == RESULT_CANCELED) {
      // Apply potentially new settings

      mSoundOn = mPrefs.getBoolean("sound", true);

      String difficultyLevel = mPrefs.getString("difficulty_level",
         getResources().getString(R.string.difficulty_harder));

      if (difficultyLevel.equals(getResources().getString(R.string.difficulty_easy)))
            mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.Easy);
      else if (difficultyLevel.equals(getResources().getString(R.string.difficulty_harder)))
            mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.Harder);
      else
            mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.Expert);
      }
   }
}
```

The mSoundOn variable from above is not yet defined. **It is left to you** to declare this class-level boolean variable and use it to determine if the mHumanMediaPlayer or mComputerMediaPlayer should be played.

5. In order for the Intent to work, you must also modify the app's AndroidManifest.xml file, adding the following inside the <application> tags which indicate the Settings activity should be included as part of this app:

```
<!-- Allow the Settings activity to be launched -->
<activity android:name=".Settings" android:label="Settings"></activity>
```

6. The sound settings and the difficulty level need to be set in AndroidTicTacToe's onCreate() method like so:

```
// Restore the scores from the persistent preference data source
mPrefs = PreferenceManager.getDefaultSharedPreferences(this);
mSoundOn = mPrefs.getBoolean("sound", true);
String difficultyLevel = mPrefs.getString("difficulty_level",
            getResources().getString(R.string.difficulty_harder));
if (difficultyLevel.equals(getResources().getString(R.string.difficulty_easy)))
   mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.Easy);
else if (difficultyLevel.equals(getResources().getString(R.string.difficulty_harder)))
   mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.Harder);
else
   mGame.setDifficultyLevel(TicTacToeGame.DifficultyLevel.Expert);
```

Note that the code segment above initializes the mPrefs variable slightly differently than was done in the previous tutorial. This is so the `PreferenceActivity` and the `AndroidTicTacToe` activity can both be using the same settings.

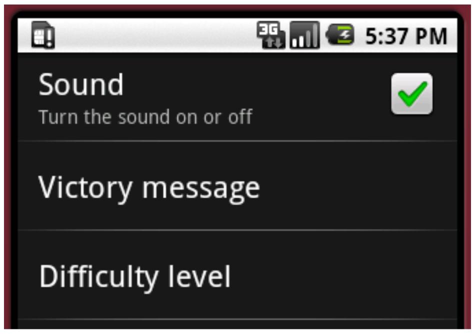
7. Finally, change the code in the `endGame()` function to display the victory message preference:

```
else if (winner == 2) {
    mHumanWins++;
    mHumanScoreTextView.setText(Integer.toString(mHumanWins));
    String defaultMessage = getResources().getString(R.string.result_human_wins);
    mInfoTextView.setText(mPrefs.getString("victory_message", defaultMessage));
}
```

Run your app and launch the Settings screen by selecting the Menu and pressing **Settings**. Verify that you can now turn the sound on and off, set the victory message, and set the difficulty level. We've gained a lot of functionality with little code.

Displaying Setting Information

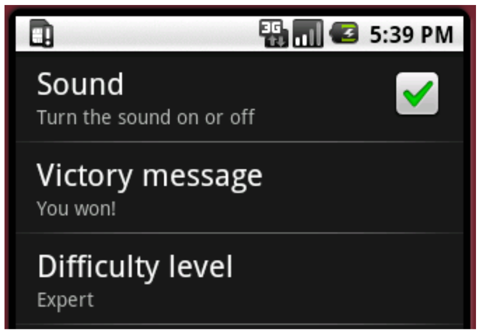
There is one thing that would make the settings a little more user friendly… displaying the actual victory message and difficulty level inside the settings menu. That way the user doesn't have to actually select the setting just to see what it is set to. In other words, we want to change the settings on the left to look like the settings on the right:


5:37 PM

Sound
Turn the sound on or off

Victory message

Difficulty level


5:39 PM

Sound
Turn the sound on or off

Victory message
You won!

Difficulty level
Expert

1. In the Settings.java file, add the following code to the `onCreate()` method:

```
final SharedPreferences prefs =
         PreferenceManager.getDefaultSharedPreferences(getBaseContext());

final ListPreference difficultyLevelPref = (ListPreference)
         findPreference("difficulty_level");
String difficulty = prefs.getString("difficulty_level",
         getResources().getString(R.string.difficulty_expert));
difficultyLevelPref.setSummary((CharSequence) difficulty);

difficultyLevelPref.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
   @Override
   public boolean onPreferenceChange(Preference preference, Object newValue) {
      difficultyLevelPref.setSummary((CharSequence) newValue);

      // Since we are handling the pref, we must save it
      SharedPreferences.Editor ed = prefs.edit();
      ed.putString("difficulty_level", newValue.toString());
      ed.commit();

      return true;
   }
});
```

The code above first sets the difficulty level's setting's summary to the currently selected difficulty level. Then it creates an `OnPreferenceChangeListener` which will execute when the difficulty level is changed. This is necessary so we can change the summary when a different level is selected, but it also makes us manually have to save the preference.

2. We'll do something similar for the victory message:

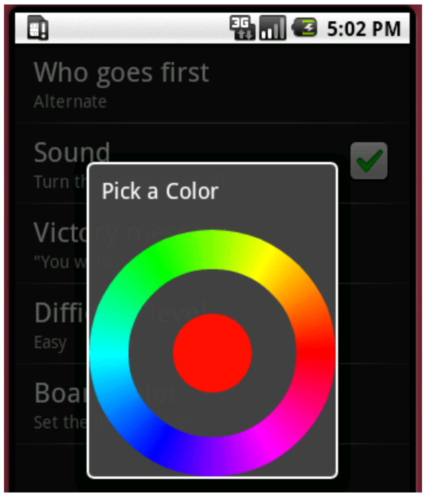
```
final EditTextPreference victoryMessagePref = (EditTextPreference)
         findPreference("victory_message");
String victoryMessage = prefs.getString("victory_message",
         getResources().getString(R.string.result_human_wins));
victoryMessagePref.setSummary("\"" + victoryMessage + "\"");
victoryMessagePref.setOnPreferenceChangeListener(new OnPreferenceChangeListener() {
   @Override
   public boolean onPreferenceChange(Preference preference, Object newValue) {
      victoryMessagePref.setSummary((CharSequence) newValue);

      SharedPreferences.Editor ed = prefs.edit();
      ed.putString("victory_message", newValue.toString());
      ed.commit();

      return true;
   }
});
```

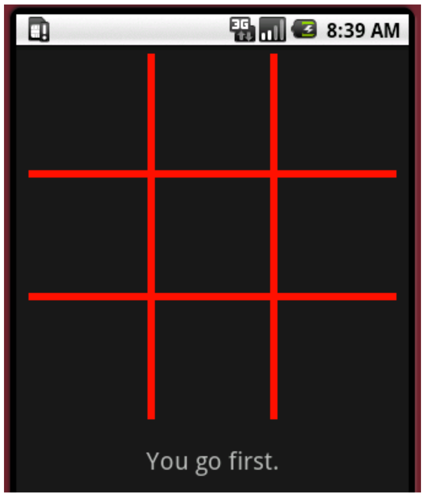
Now run the app again, and this time verify that the settings now display the current victory message and difficulty level.

Extra Challenge

Create a **Board color** preference that will allow the user to change the board color. An idea method is to display the `ColorPickerDialog` that is provided in the Android API demos: http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/ColorPickerDialog.html

As shown in the figure below, the user clicks an area around the circle to pick a color and then must click the circle in the middle to dismiss the dialog. When they return to the game, the board color should match the selected color.





The `Settings` class will need to instantiate the `ColorPickerDialog` using the current color setting. The constructor takes a `OnColorChangedListener` which is used to notify the caller when the color is selected. You will need to save the selected color in the same manner in which you saved other preferences. The color should be applied to the `BoardView's` board color. You will need to create getters and setters for the board color in the `BoardView` to accomplish this task.