Module 3 : UI Design and Data storage

Saving to External Storage (SD Card)

The previous section showed how you can save your files to the internal storage of your Android device. Sometimes, it would be useful to save them to external storage (such as an SD card) because of its larger capacity, as well as the capability to share the files easily with other users (by removing the SD card and passing it to somebody else).

Using the project created in the previous section as the example, to save the text entered by the user in the SD card, modify the onClick() method of the Save button as shown in bold here:

```
saveBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
String str = textBox.getText().toString();
try
{
//---SD Card Storage---
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +
"/MyFiles");
directory.mkdirs();
File file = new File(directory, "textfile.txt");
FileOutputStream fOut = new FileOutputStream(file);
OutputStreamWriter osw = new
OutputStreamWriter(fOut);
//---write the string to the file---
osw.write(str);
osw.flush();
osw.close();
//---display file saved message---
Toast.makeText(getBaseContext(),
"File saved successfully!",
Toast.LENGTH_SHORT).show();
//---clears the EditText---
textBox.setText("");
}
```

```
catch (IOException ioe)
{
ioe.printStackTrace();
}
}
});
```

The preceding code uses the getExternalStorageDirectory() method to return the full path to the external storage. Typically, it should return the "/sdcard" path for a real device, and "/mnt/sdcard" for an Android Emulator. However, you should never try to hardcode the path to the SD card, as manufacturers may choose to assign a different path name to the SD card. Hence, be sure to use the getExternalStorageDirectory() method to return the full path to the SD card.

You then create a directory called MyFiles in the SD card. Finally, you save the file into this directory.

To load the file from the external storage, modify the onClick() method for the Load button:

```
loadBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
try
{
//---SD Storage---
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +
"/MyFiles");
File file = new File(directory, "textfile.txt");
FileInputStream fIn = new FileInputStream(file);
InputStreamReader isr = new InputStreamReader(fIn);
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{
//---convert the chars to a String---
String readString =
String.copyValueOf(inputBuffer, 0, charRead);
s += readString;
inputBuffer = new char[READ_BLOCK_SIZE];
```

```
}
//---set the EditText to the text that has been
// read---
textBox.setText(s);
Toast.makeText(getBaseContext(),
"File loaded successfully!",
Toast.LENGTH_SHORT).show();
}
catch (IOException ioe) {
ioe.printStackTrace();
}
}
});
```

Note that in order to write to the external storage, you need to add the WRITE_EXTERNAL_STORAGE permission in your AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="net.learn2develop.Files"
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon" android:label="@string/app_name">
<activity android:name=".MainActivity"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
<uses-sdk android:minSdkVersion="9" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
</uses-permission>
</manifest>
```

Choosing the Best Storage Option

And so, you have seen the few ways of saving data in your Android applications — SharedPreferences, internal storage, and external storage. Which one should you use in your applications? Here are some suggestions:

➤➤ If you have data that can be represented using key/value pairs, then use the SharedPreferences object. For example, if you want to store user preference data such as user name, background color, date of birth, last login date, then the SharedPreferences object is the ideal way to store these data. Moreover, you don't really have to get your hands dirty on how these data are stored; all you need is to use the SharedPreferences object to store and retrieve them.

➤➤ If you need to store ad-hoc data, then using the internal storage is a good option. For example, your application (such as an RSS reader) may need to download images from the Web for display. In this scenario, saving the images to internal storage is a good solution. You may also need to persist data created by the user, such as when you have a note-taking application where users can take notes and save them for later use. In all these scenarios, using the internal storage is a good choice.

➤➤ There are times when you need to share your application data with other users. For example, you may create an Android application that logs the coordinates of the locations that a user has been to, and you want to share all these data with other users. In this scenario, you can store your files on the SD card of the device so that users can easily transfer the data to other devices (and computers) for use later.

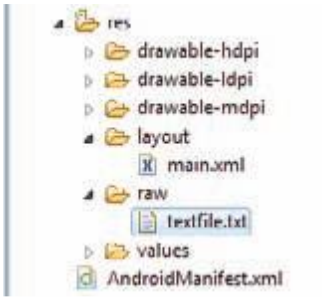
Using Static Resources

Besides creating and using files dynamically during run time, it is also possible to add files to your package during design time so that you can use it during run time. For example, you may want to bundle some help files with your package so that you can display some help messages when users need it. In this case, you can add the files to the res/raw folder (you need to create this folder yourself) of your package. Figure 6-8 shows the res/raw folder containing a file named textfile.txt. To make use of the file in code, use the getResources() method to return a Resources object and then use its openRawResource() method to open the file contained in the res/raw folder:

```
import java.io.InputStream;
import java.io.BufferedReader;
/** Called when the activity is first created. */
@Override
```

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
InputStream is = this.getResources().openRawResource(R.raw.textfile);
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String str = null;
try {
while ((str = br.readLine()) != null) {
Toast.makeText(getBaseContext(),
str, Toast.LENGTH_SHORT).show();
}
is.close();
br.close();
} catch (IOException e) {
e.printStackTrace();
}
textBox = (EditText) findViewById(R.id.txtText1);
Button saveBtn = (Button) findViewById(R.id.btnSave);
Button loadBtn = (Button) findViewById(R.id.btnLoad);
saveBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
}
});
loadBtn.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
}
});
}
```

The resource ID of the resource stored in the res/raw folder is named after its filename without its extension. For example, if the text file is textfile.txt, then its resource ID is R.raw.textfile.


res
drawable-hdpi
drawable-ldpi
drawable-mdpi
layout
main.xml
raw
textfile.txt
values
AndroidManifest.xml