CMPS 121
Preferences and files

Luca de Alfaro

Portions of this presentations are based on content taken from the Android Developers Site. The work was created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.
SharedPreferences

Framework for saving key-value pairs of primitive data: strings, floats, booleans, ints, longs. To get a SharedPreferences object, use one of:

- `getSharedPreferences()`
- `getDefaultSharedPreferences()`

To write:

- Call `edit()` to get a `SharedPreferences.Editor`
- `putBoolean()`, `putString()`, `...`, `commit()` on the editor

To read:

- `getBoolean()`, `getString()`
Example

```java
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state) {
        super.onCreate(state);
        ... 

        // Restore preferences
        SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
        boolean silent = settings.getBoolean("silentMode", false);
        setSilent(silent);
    }
```
Example (cont.)

```java
@Override
protected void onStop(){
    super.onStop();

    // We need an Editor object to make preference changes.
    // All objects are from android.context.Context
    SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
    SharedPreferences.Editor editor = settings.edit();
    editor.putBoolean("silentMode", mSilentMode);

    // Commit the edits!
    editor.commit();
}
```
Preference activity framework

1. Preference screen layout
2. Preference activity
3. Shared preferences change listener
Preference layout in XML

<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory
        android:title="My Preference Category"/>
    <CheckBoxPreference
        android:key="PREF_CHECK_BOX"
        android:title="Check Box Preference"
        android:summary="Check Box Preference Description"
        android:defaultValue="true"/>
</PreferenceCategory>
</PreferenceScreen>
Preference layout in XML

Preference controls:
- CheckBoxPreference
- EditTextPreference
- ListPreference. List of values from which to select
- RingTonePreference

See documentation for the Preference class in the Android developers center.

Note that the file needs to be in res/xml and not in the layouts folder.
package com.paad.earthquake;

import ...
public class Preferences extends PreferenceActivity {
    public static final String PREF_AUTO_UPDATE = "PREF.AUTO_UPDATE";
    ...
    SharedPreferences prefs;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        super.onCreate(icicle);
        addPreferencesFromResource(R.xml.userpreferences);
    }
}

Accessing preferences

To access preferences from any place:

```java
Context context = getApplicationContext();
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
// Retrieve preferences with prefs.get_<type> methods
```
Preferences import/export

You can use intents to import preferences from other applications (e.g., display preferences...).

You can also make your preferences available to other applications.

You can respond to changes in preferences by implementing a Shared Preferences Change Listener (see next slide).
Shared Preferences Change Listener

public class MyActivity extends Activity implements OnSharedPreferenceChangeListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        // Register this OnSharedPreferenceChangeListener
        Context context = getApplicationContext();
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
        prefs.registerOnSharedPreferenceChangeListener(this);
    }

    public void onSharedPreferenceChanged(SharedPreferences prefs, String key) {
        // TODO Check the shared preference and key parameters and change UI or behavior as appropriate.
    }
}

Using the Internal Storage

Default: files created by an app are private to the app, and are removed when the app is uninstalled.

Apps correspond to users in linux, permission-wise.

To create a file:

- Call openFileOutput() with the name of the file
- Write to the file with write(), then close().

```java
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```
Using the Internal Storage

- Call openFileInput().
- Read bytes with read(byte[] buffer, int byteOffset, int byteCount),
- Then close with close().
Using the External Storage (SD card)

- No security enforced (including for photos!)
- Can change without warning if the user unmounts, etc.
Check external storage availability

```java
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states,
    // but all we need to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```
Accessing external files

In API 8 or greater:

`getExternalFilesDir(type)` returns a place on the external file system that is private to the application. When the application is uninstalled, these files are deleted. But:

- Cards can be removed
- There is no security.

`type`, if not null, can be:

```
DIRECTORY_MUSIC, DIRECTORY_PODCASTS,
DIRECTORY_RINGTONES, DIRECTORY_ALARMS,
DIRECTORY_NOTIFICATIONS, DIRECTORY_PICTURES,
DIRECTORY_MOVIES
```
Example

```java
void createExternalStoragePrivateFile() {
    // Create a path where we will place our private file on external
    // storage.
    File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
    try {
        // Very simple code to copy a picture from the application's
        // resource into the external file. Note that this code does
        // no error checking, and assumes the picture is small (does not
        // try to copy it in chunks). Note that if external storage is
        // not currently mounted this will silently fail.
        InputStream is = getResources().openRawResource(R.drawable.balloons);
        OutputStream os = new FileOutputStream(file);
        byte[] data = new byte[is.available()];
        is.read(data);
        os.write(data);
        is.close();
        os.close();
    } catch (IOException e) {
        // Unable to create file, likely because external storage is
        // not currently mounted.
        Log.w("ExternalStorage", "Error writing " + file, e);
    }
}
```
Example (cont.)

```java
void deleteExternalStoragePrivateFile() {
    // Get path for the file on external storage. If external
    // storage is not currently mounted this will fail.
    File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
    if (file != null) {
        file.delete();
    }
}

boolean hasExternalStoragePrivateFile() {
    // Get path for the file on external storage. If external
    // storage is not currently mounted this will fail.
    File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
    if (file != null) {
        return file.exists();
    }
    return false;
}
```
You can connect a file to the media scanner

```java
File file = new File(path, "DemoPicture.jpg");

try {
    // ...somehow write the file (see previous example)...

    // Tell the media scanner about the new file so that it is
    // immediately available to the user.
    MediaScannerConnection.scanFile(this,
        new String[] { file.toString() }, null,
        new MediaScannerConnection.OnScanCompletedListener() {
            public void onScanCompleted(String path, Uri uri) {
                Log.i("ExternalStorage", "Scanned " + path + ":");
                Log.i("ExternalStorage", "-> uri=" + uri);
            }
        });
```
Accessing "public" external files

You can connect these files to the media scanner using MediaScannerConnection.scanFile.

Note, however, that the directory returned by getExternalFilesDir is private to the app, and will be deleted when the app is deleted.
Example of connection to media scanner

```java
void createExternalStoragePrivatePicture() {
    // Create a path where we will place our picture in our own private
    // pictures directory. Note that we don't really need to place a
    // picture in DIRECTORY_PICTURES, since the media scanner will see
    // all media in these directories; this may be useful with other
    // media types such as DIRECTORY_MUSIC however to help it classify
    // your media for display to the user.
    File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    File file = new File(path, "DemoPicture.jpg");
}```
Example (cont.)

```java
try {
    // Very simple code to copy a picture from the application's
    // resource into the external file. Note that this code does
    // no error checking, and assumes the picture is small (does not
    // try to copy it in chunks). Note that if external storage is
    // not currently mounted this will silently fail.
    InputStream is = getResources().openRawResource(R.drawable.balloons);
    OutputStream os = new FileOutputStream(file);
    byte[] data = new byte[is.available()];
    is.read(data);
    os.write(data);
    is.close();
    os.close();
    // Tell the media scanner about the new file so that it is
    // immediately available to the user.
    MediaScannerConnection.scanFile(this,
        new String[] { file.toString() }, null,
        new MediaScannerConnection.OnScanCompletedListener() {
            public void onScanCompleted(String path, Uri uri) {
                Log.i("ExternalStorage", "Scanned " + path + ":");
                Log.i("ExternalStorage", "-> uri=" + uri);
            }
        });
```
Example (cont.)

```java
} catch (IOException e) {
    // Unable to create file, likely because external storage is
    // not currently mounted.
    Log.w("ExternalStorage", "Error writing " + file, e);
}

void deleteExternalStoragePrivatePicture() {
    // Create a path where we will place our picture in the user's
    // public pictures directory and delete the file. If external
    // storage is not currently mounted this will fail.
    File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    if (path != null) {
        File file = new File(path, "DemoPicture.jpg");
        file.delete();
    }
}
Example (cont.)

```java
boolean hasExternalStoragePrivatePicture() {
    // Create a path where we will place our picture in the user's
    // public pictures directory and check if the file exists. If
    // external storage is not currently mounted this will think the
    // picture doesn't exist.
    File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    if (path != null) {
        File file = new File(path, "DemoPicture.jpg");
        return file.exists();
    }
    return false;
}
```
Getting a public file

To get a directory for public files, use instead `Environment.getExternalStoragePublicDirectory()`

Example:

```java
void createExternalStoragePublicPicture() {
    // Create a path where we will place our picture in the user's
    // public pictures directory. Note that you should be careful about
    // what you place here, since the user often manages these files. For
    // pictures and other media owned by the application, consider
    // Context.getExternalMediaDir().
    File path = Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES);
    File file = new File(path, "DemoPicture.jpg");
}```