

## Lưu trữ, điều chỉnh và phục hồi các thiết lập của người dùng trong một ứng dụng

Một ứng dụng thường bao gồm các **thiết lập** (setting) cho phép người dùng thay đổi các tính năng hoặc hoạt động của ứng dụng. Ví dụ, vài ứng dụng cho phép người dùng thiết lập sự xuất hiện các thông báo (notifications) là có hay không; xác định cách ứng dụng đồng bộ dữ liệu với điện toán đám mây; chọn phong chữ, hình nền; khai báo và lưu thông tin đăng nhập (tên đăng nhập, mật mã,...) của người dùng để sau này mở lại không cần phải khai báo;... Nếu người lập trình muốn xây dựng các **thiết lập tùy chỉnh** (custom setting, preference) cho ứng dụng của mình thì nên sử dụng API Preference. Thay vì sử dụng các đối tượng View như trong xây dựng giao diện người dùng, các thiết lập này được xây dựng bằng các lớp con của lớp Preference và chúng được khai báo trong một **tập tin XML**.

Một đối tượng Preference được dùng để xây dựng một thiết lập. Mỗi Preference xuất hiện như là một mục (item) trong một danh sách và cung cấp giao diện thích hợp cho người dùng chỉnh sửa các thiết lập. Ví dụ, một CheckBoxPreference tạo ra một checkbox để chọn lựa các thiết lập, một ListPreference tạo ra một mục mà nó cho phép mở ra một hộp thoại với một danh sách các lựa chọn, một EditTextPreference mở ra một hộp thoại (dialog) cho người dùng nhập vào một văn bản và lưu giá trị của nó như là một String.

Mỗi Preference mà người lập trình thêm vào tương ứng với một cặp **key-value** mà hệ thống dùng để lưu một thiết lập trong một tập tin SharedPreferences mặc định. Khi người thay đổi một thiết lập, hệ thống cập nhật value tương ứng trong tập tin SharedPreferences. Chúng ta cần tương tác trực tiếp với tập tin SharedPreferences có liên quan chỉ khi chúng ta cần đọc các value để xác định hoạt động của ứng dụng trên thiết lập của người dùng. Các value được lưu trong SharedPreferences cho mỗi thiết lập có thể lấy một trong các giá trị sau: Boolean, Float, Int, Long, String, String set.

Bởi vì các thiết lập của một ứng dụng được xây dựng bằng đối tượng Preference thay vì đối tượng View, nên ta phải dùng một Activity hay Fragment đặc biệt để hiển thị danh sách các thiết lập:

- Nếu ứng dụng được hỗ trợ bởi phiên bản cũ hơn Android 3.0 (tương ứng API 10 hay thấp hơn), ta phải xây dựng Activity kế thừa của lớp PreferenceActivity.
- Nếu ứng dụng hỗ trợ bởi Android 3.0 về sau, ta nên sử dụng Activity truyền thống với một PreferenceFragment để hiển thị các thiết lập của ứng dụng. Tuy nhiên chúng ta cũng có thể sử dụng PreferenceActivity để tạo ra một layout hai của sổ

(two-pane Layout) cho các thiết bị có màn hình rộng, khi chúng ta có nhiều nhóm thiết lập.

## 1. Định nghĩa một Preference bằng XML

Mặc dù ta có thể tạo ra một đối tượng Preference mới trong thời gian chạy, nhưng ta nên định nghĩa danh sách các thiết lập trong XML với một hệ thống thứ bậc các đối tượng Preference. Việc sử dụng một tập tin XML để định nghĩa một tập các thiết lập được ưa chuộng hơn bởi vì tập tin XML cung cấp một cấu trúc dễ đọc và dễ dàng cập nhật. Các thiết lập của một ứng dụng cũng thường được xác định trước, và ta cũng có thể thay đổi bộ thiết lập này trong thời gian chạy.

Mỗi lớp con của Preference được khai báo với một phần tử XML trùng với tên lớp, ví dụ như <CheckBoxPreference>.

Chúng ta phải lưu tập tin XML vào thư mục **res/xml/**. Mặc dù ta có thể đặt tên tập tin với bất kỳ tên nào mà ta muốn, nhưng tên truyền thống của nó là `preference.xml`. Chúng ta thường chỉ dùng một tập tin, bởi vì các nhánh trong hệ thống thứ bậc (mà nó mở danh sách các thiết lập) được khai báo bằng cách dùng các **thể hiện** (instance) lồng vào nhau của PreferenceScreen.

Lưu ý: nếu muốn tạo ra một layout nhiều cửa sổ, thì ta phải khai báo các tập tin XML khác nhau cho mỗi fragment.

Nút gốc của tập tin XML phải là một phần tử <PreferenceScreen>. Bên trong các phần tử này là nơi để ta thêm vào các đối tượng Preference. Mỗi đối tượng được thêm vào trong phần tử <PreferenceScreen> sẽ xuất hiện như một mục trong danh sách các thiết lập.

### Ví dụ:

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

<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="pref_sync"
        android:title="@string/pref_sync"
        android:summary="@string/pref_sync_summ"
        android:defaultValue="true" />
    <ListPreference
        android:dependency="pref_sync"
```

```
android:key="pref_syncConnectionType"
android:title="@string/pref_syncConnectionType"
android:dialogTitle="@string/pref_syncConnectionType"
    android:entries="@array/pref_syncConnectionTypes_entries"
    android:entryValues="@array/pref_syncConnectionTypes_values"
    android:defaultValue="@string/pref_syncConnectionTypes_default" />
</PreferenceScreen>
```

Trong ví dụ này, có một CheckBoxPreference và một ListPreference. Cả hai mục có 3 thuộc tính sau:

**android:key**, thuộc tính này được đòi hỏi cho preference mà nó luôn giữ một giá trị dữ liệu. Nó xác định một **key** duy nhất (một string) mà hệ thống sử dụng khi lưu giá trị của thiết lập này trong SharedPreferences. Thuộc tính này không được đòi hỏi chỉ khi preference là một PreferenceCategory hay PreferenceScreen, hoặc preference xác định một Intent (với một phần tử <intent> ) hoặc một Fragment (với một thuộc tính android:fragment ).

**android:title**, thuộc tính này cung cấp một tên mà người dùng nhìn thấy cho một thiết lập.

**android:defaultValue**, thuộc tính này là giá trị ban đầu mà hệ thống cần đặt trong tập tin SharedPreferences . Ta nên cung cấp giá trị mặc định cho tất cả các thiết lập.

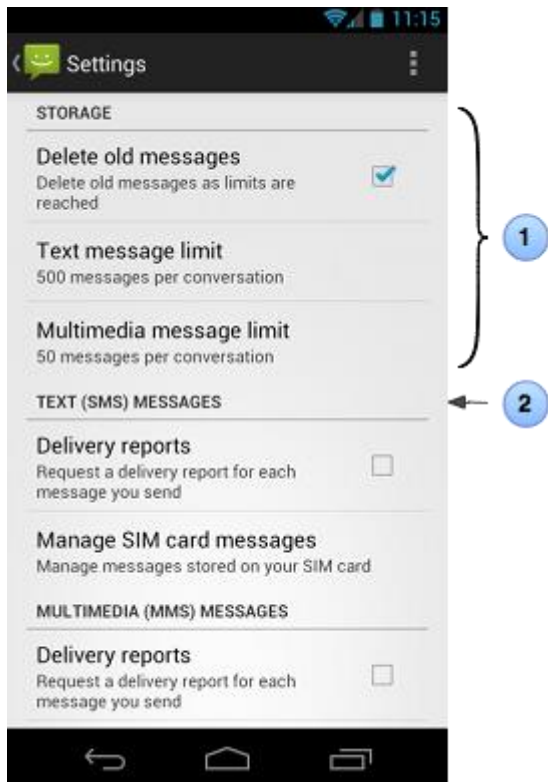
Khi danh sách các thiết lập vượt hơn 10 mục, Ta có thể thêm các tiêu đề để xác định các nhóm thiết lập hoặc hiển thị các nhóm này trong một màn hình riêng biệt. Các tùy chọn này được mô tả trong các phần sau.

## 2. Tạo ra một nhóm thiết lập

Nếu ta trình bày một danh sách từ 10 thiết lập trở lên, người dùng có thể gặp khó khăn khi xem xét, tìm hiểu, và xử lý chúng. Ta có thể khắc phục điều này bằng cách chia một số hoặc tất cả các thiết lập thành các nhóm, sự chuyển một danh sách dài thành nhiều danh sách ngắn hơn sẽ thuận lợi hơn. Một nhóm các thiết lập liên quan nhau có thể được thể hiện bằng một trong hai cách sau: *dùng các tiêu đề* hoặc *dùng các màn hình con*. Ta có thể dùng một hay cả hai kỹ thuật gom nhóm này để tổ chức các thiết lập của một ứng dụng.

### 2.1. Dùng tiêu đề

Nếu chúng ta muốn dùng các tiêu đề chen vào giữa các nhóm thiết lập để phân nhóm (hình 5.1) thì đặt các nhóm Preference bên trong một PreferenceCategory.



**Hình 5.1:** Phân loại thiết lập với các tiêu đề

- (1) Loại thiết lập được phân chia bởi phần tử **<PreferenceCategory>**.
- (2) Tiêu đề được xác định bởi thuộc tính **android:title**

**Ví dụ:**

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

    <PreferenceCategory
        android:title="@string/pref_sms_storage_title"
        android:key="pref_key_storage_settings">

        <CheckBoxPreference
            android:key="pref_key_auto_delete"
            android:summary="@string/pref_summary_auto_delete"
            android:title="@string/pref_title_auto_delete"
            android:defaultValue="false" ... />

        <Preference
            android:key="pref_key_sms_delete_limit"
            android:dependency="pref_key_auto_delete"
            android:summary="@string/pref_summary_delete_limit"
            android:title="@string/pref_title_sms_delete" ... />

        <Preference
            android:key="pref_key_mms_delete_limit"
```

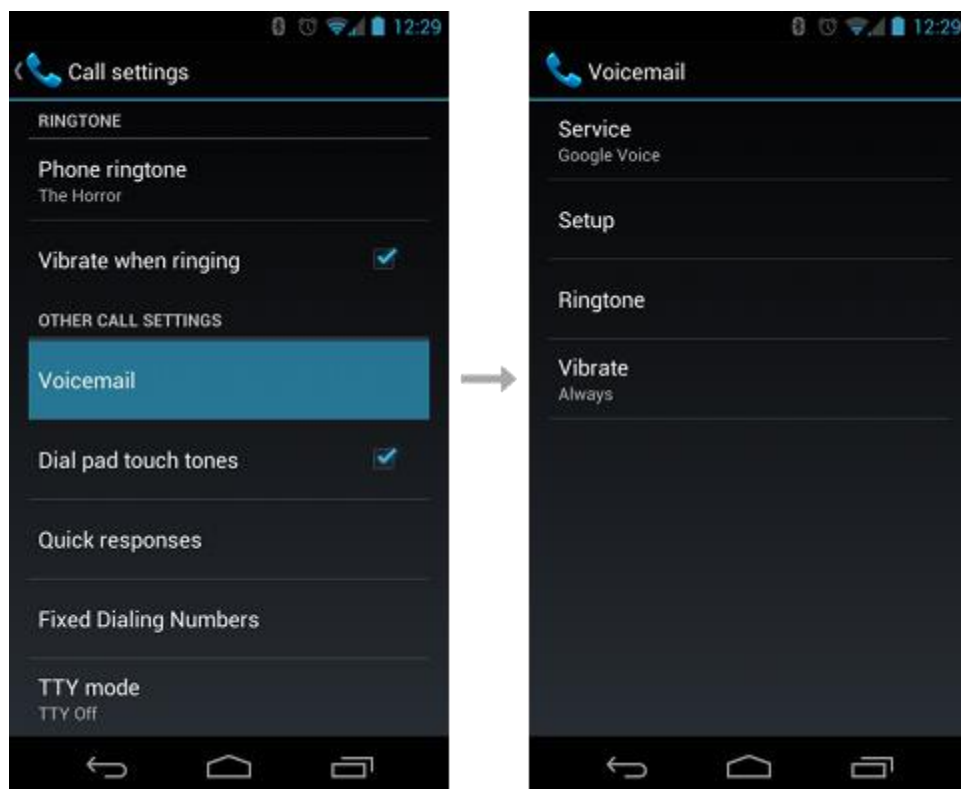
```

        android:dependency="pref_key_auto_delete"
        android:summary="@string/pref_summary_delete_limit"
        android:title="@string/pref_title_mms_delete" ... />
    </PreferenceCategory>
    ...
</PreferenceScreen>

```

## 2.2. Dùng màn hình con

Nếu muốn đặt các nhóm thiết lập vào một màn hình con (Hình 5.2) thì ta đặt các nhóm đối tượng [Preference](#) vào bên trong [PreferenceScreen](#).



**Hình 5.2:** Màn hình con trình bày các thiết lập

Trong hình 5.2 phần tử <PreferenceScreen> tạo ra một mục, khi chọn vào mục này, một màn hình con được mở ra hiển thị danh sách thiết lập có tiêu đề là Voicemail.

**Ví dụ:**

```

<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- opens a subscreen of settings -->
  <PreferenceScreen
    android:key="button_voicemail_category_key"
    android:title="@string/voicemail"
    android:persistent="false">
    <ListPreference
      android:key="button_voicemail_provider_key"
      android:title="@string/voicemail_provider" ... />
    <!-- opens another nested subscreen -->
    <PreferenceScreen
      android:key="button_voicemail_setting_key"
      android:title="@string/voicemail_settings"
      android:persistent="false">
      ...
    </PreferenceScreen>
    <RingtonePreference
      android:key="button_voicemail_ringtone_key"
      android:title="@string/voicemail_ringtone_title"
      android:ringtoneType="notification" ... />
    ...
  </PreferenceScreen>
  ...
</PreferenceScreen>

```

### 2.3. Dùng intents

Trong vài trường hợp, ta muốn có một mục preference để mở một activity khác thay vì một màn hình thiết lập, chẳng hạn như mở một trình duyệt web để xem một trang web. Để gọi một intent khi người dùng chọn một mục preference, ta thêm vào một phần tử <intent> như là con của phần tử <Preference> tương ứng.

Ví dụ, sau đây là cách dùng một mục preference để mở một trang web:

```

<Preference android:title="@string/prefs_web_page" >
  <intent android:action="android.intent.action.VIEW"
    android:data="http://www.example.com" />
</Preference>

```

Ta có thể tạo ra cả hai intent không tường minh và intent tường minh. Bằng cách dùng các thuộc tính sau:

**android:action**

hành động được chỉ định, tác dụng như phương thức [setAction\(\)](#) .

**android:data**

dữ liệu được chỉ định, tác dụng như phương thức [setData\(\)](#) .

**android:mimeType**

loại MIME được chỉ định, tác dụng như phương thức [setType\(\)](#).

**android:targetClass**

Class của thành phần của ứng dụng được chọn để điều khiển intent, tác dụng như phương thức [setComponent\(\)](#).

**android:targetPackage**

Package của thành phần của ứng dụng được chọn để điều khiển intent, tác dụng như phương thức [setComponent\(\)](#) .

### 3. Tạo ra một Preference Activity

Để hiển thị các thiết lập trong một activity, ta sử dụng lớp [PreferenceActivity](#). Đây là sự mở rộng của lớp activity truyền thống mà nó hiển thị một danh sách các thiết lập dựa trên hệ thống thứ bậc của các đối tượng. Lớp [PreferenceActivity](#) tự động duy trì các thiết lập liên quan với mỗi [Preference](#) khi người dùng thực hiện một sự thay đổi.

**Ghi chú:** Nếu chúng ta phát triển ứng dụng cho Android 3.0 hoặc cao hơn, ta nên dùng [PreferenceFragment](#) thay vì [PreferenceActivity](#).

Điều quan trọng cần nhớ là ta không tải layout của các views bằng phương thức [onCreate\(\)](#), thay vào đó, ta dùng phương thức [addPreferencesFromResource\(\)](#) để thêm preferences mà chúng ta đã khai báo trong tập tin XML vào activity.

Ví dụ: Sau đây là mã tối thiểu cho một [PreferenceActivity](#):

```
public class SettingsActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

Đoạn lệnh trên là đủ cho một số ứng dụng, ngay khi người dùng thay đổi một preference, hệ thống sẽ lưu những thay đổi vào một tập tin [SharedPreferences mặc định](#) mà các thành phần ứng dụng khác có thể đọc khi ta cần kiểm tra các thiết lập của người dùng. Tuy

nhiên, trong một số ứng dụng đòi hỏi phải viết thêm một ít mã để lắng nghe những thay đổi trong các preferences.

## Dùng PreferenceFragments

Nếu chúng ta phát triển ứng dụng cho Android 3.0 (API level 11) hoặc cao hơn, ta nên dùng [PreferenceFragment](#) để hiển thị danh sách các đối tượng [Preference](#). Ta có thể thêm một [PreferenceFragment](#) vào một activity bất kỳ mà không cần dùng [PreferenceActivity](#).

[Fragments](#) cung cấp một kiến trúc linh động cho ứng dụng so với trường hợp chỉ dùng activitie. Chẳng hạn, chúng ta dùng [PreferenceFragment](#) để điều khiển hiển thị các thiết lập thay cho [PreferenceActivity](#) khi có thể.

Việc thực thi [PreferenceFragment](#) đơn giản như định nghĩa hàm [onCreate\(\)](#) để tải một tập tin preferences với [addPreferencesFromResource\(\)](#).

### Ví dụ:

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.preferences);
    }
    ...
}
```

Sau đó chúng ta có thể thêm fragment này và một [Activity](#) và giống như vậy cho bất kỳ các [Fragment](#) khác.

### Ví dụ:

```
public class SettingsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Display the fragment as the main content.
        getFragmentManager().beginTransaction()
```



```
        .replace(android.R.id.content, new SettingsFragment())
        .commit();
    }
}
```

**Chú ý:** Một [PreferenceFragment](#) không có đối tượng [Context](#) của riêng nó. Nếu ta cần có một đối tượng [Context](#), ta có thể gọi hàm [getActivity\(\)](#). Tuy nhiên, ta chỉ gọi hàm [getActivity\(\)](#) khi fragment gắn với một activity. Khi fragment chưa được gắn hay đã được gỡ bỏ cho đến cuối vòng đời của activity thì hàm [getActivity\(\)](#) sẽ trả về null.

#### 4. Thiết lập giá trị mặc định

Preferences có thể định nghĩa một số hoạt động cho ứng dụng, vì vậy cần thiết phải khởi động một tập tin liên kết [SharedPreferences](#) với các giá trị mặc định cho mỗi [Preference](#) khi người dùng mở ứng dụng lần đầu tiên.

Trước tiên ta phải xác định giá trị mặc định cho mỗi đối tượng trong tập tin XML bằng cách dùng thuộc tính `android:defaultValue`. Giá trị này có thể là một loại dữ liệu nào đó mà nó phù hợp với đối tượng [Preference](#) tương ứng.

##### Ví dụ:

```
<!-- default value is a boolean -->
<CheckBoxPreference
    android:defaultValue="true"
    ... />

<!-- default value is a string -->
<ListPreference
    android:defaultValue="@string/pref_syncConnectionTypes_default"
    ... />
```

Sau đó, từ hàm [onCreate\(\)](#) trong activity chính của ứng dụng, và trong activity bất kỳ khác, mà qua đó người dùng có thể nhập vào ứng dụng khi gọi hàm [setDefaultValues\(\)](#):

```
PreferenceManager.setDefaultValues(this, R.xml.advanced_preferences, false);
```

Việc gọi hàm [setDefaultValues\(\)](#) thông qua hàm [onCreate\(\)](#) bảo đảm rằng ứng dụng được khởi động tương thích với các thiết lập mặc định, mà ứng dụng cần đọc để xác định một vài hoạt động (chẳng hạn như có tải dữ liệu trên mạng viễn thông hay không).

Phương thức này có 3 đối số:

- **Context** của ứng dụng.
- **Resource ID** cho tập tin preference XML, cho những gì mà ta muốn thiết lập giá trị mặc định.
- **Một giá trị boolean** chỉ ra có hay không giá trị mặc định cần được thiết lập nhiều hơn một lần.

Khi là false, thì hệ thống sẽ thiết lập giá trị mặc định chỉ nếu phương thức này chưa bao giờ được gọi (hay **KEY\_HAS\_SET\_DEFAULT\_VALUES** trong tập tin default value shared preferences là false).

Nếu chúng ta đặt đối số thứ ba là false, ta có thể gọi phương thức này một cách an toàn ở mọi thời điểm mà activity khởi động với không có sự ghi đè lên các preferences của người dùng đã được lưu bằng cách đặt lại giá trị mặc định cho chúng. Tuy nhiên, nếu ta đặt là true, ta sẽ ghi đè lên bất kỳ giá trị trước đó bằng giá trị mặc định.

## 5. Dùng Preference Headers

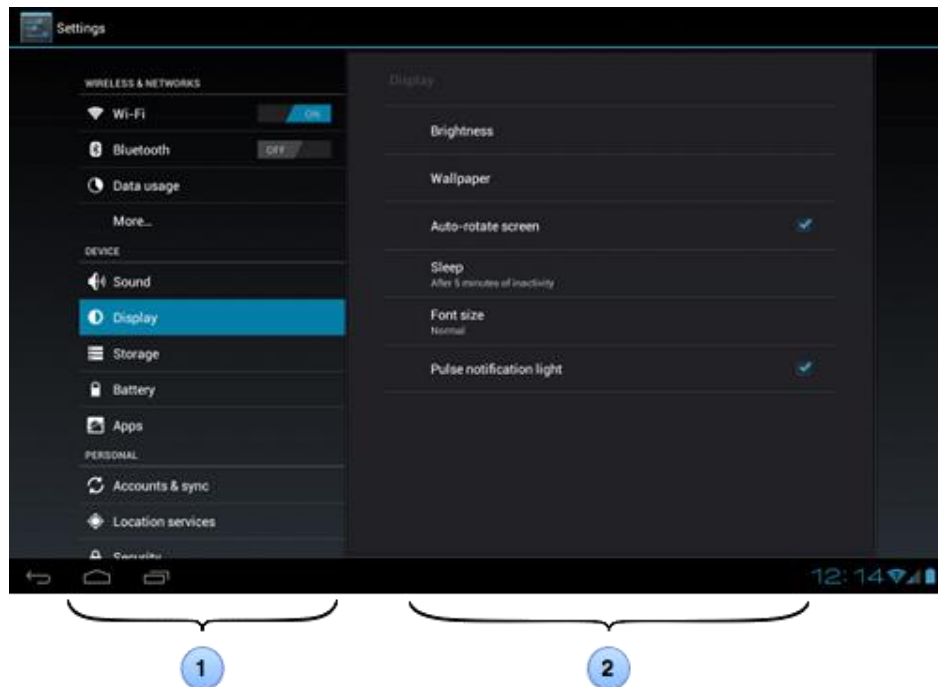
Trong một ít trường hợp, chúng ta muốn thiết kế các thiết lập đặc biệt như màn hình hiển thị đầu tiên chỉ bao gồm một danh sách các màn hình con (**subscreens**), xem minh hoạt ở hình 5.3. Khi chúng ta phát triển ứng dụng cho Android 3.0 hoặc cao hơn, chúng ta sẽ dùng tính năng "headers" mới trong Android 3.0, thay vì xây dựng các màn hình con với các phần tử **PreferenceScreen** lồng vào nhau.

Để xây dựng các thiết lập với header, chúng ta cần:

- Tách mỗi nhóm thiết lập vào những thẻ hiện (instance) của **PreferenceFragment**. Mỗi nhóm thiết lập cần một tập tin XML riêng.
- Tạo ra một tập tin XML headers, tập tin này lập danh sách mỗi nhóm thiết lập và khai báo fragment chứa danh sách tương ứng của các thiết lập.
- Thành lập lớp kế thừa của lớp **PreferenceActivity** để lưu trữ các thiết lập.
- Gọi hàm **onBuildHeaders()** để xác định tập tin headers.

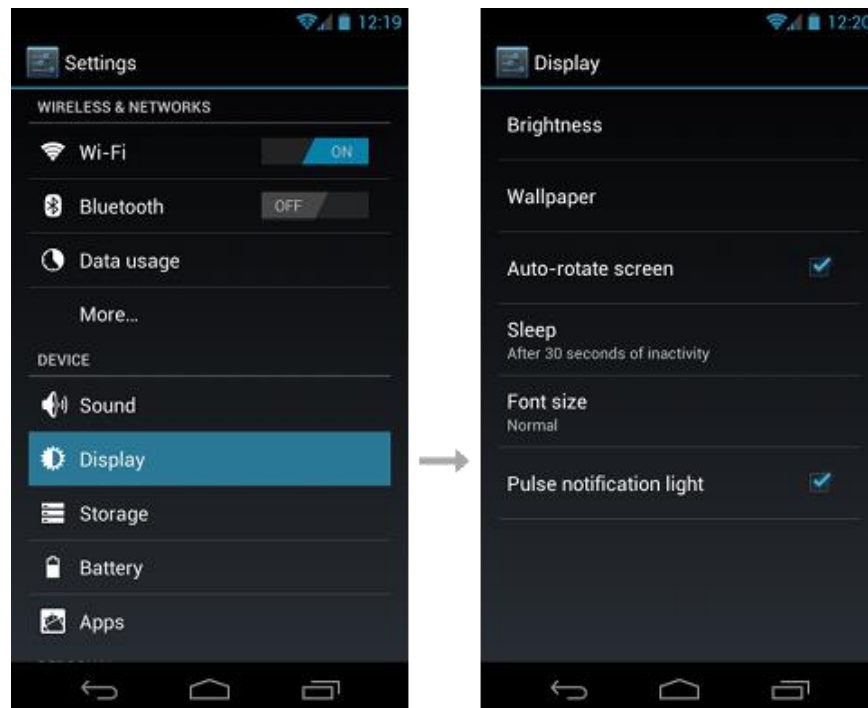
Một thuận lợi lớn trong việc dùng thiết kế này là lớp **PreferenceActivity** tự động trình bày giao diện hai cửa sổ như hình 5.4 khi ứng dụng chạy trên thiết bị có màn hình lớn.

Ngay cả các ứng dụng hỗ trợ cho các phiên bản cũ hơn 3.0, chúng ta vẫn có thể xây dựng ứng dụng dùng **PreferenceFragment** cho sự trình bày hai màn hình như trên các thiết bị đời mới, trong khi vẫn hỗ trợ hệ thống thứ bậc đa màn hình truyền thống trên các thiết bị đời cũ.



**Hình 5.4.** Giao diện 2 màn hình với header.

1. Headers được định nghĩa bằng một tập tin XML
2. Mỗi nhóm thiết lập được định nghĩa bởi [PreferenceFragment](#) mà nó được xác định bởi một phần tử <header> trong tập tin headers.



**Hình 5.5:** Thiết bị cầm tay với thiết lập header. Khi một mục được chọn, [PreferenceFragment](#) liên kết với nó thay thế header.

### Tạo ra một tập tin header

Mỗi nhóm thiết lập trong danh sách headers được xác định bởi một phần tử <header> bên trong phần tử gốc <preference-headers>.

**Ví dụ:**

```
<?xml version="1.0" encoding="utf-8"?>
<preference-headers xmlns:android="http://schemas.android.com/apk/res/android">
  <header    android:fragment="com.example.prefs.SettingsActivity$SettingsFragment
One"
    android:title="@string/prefs_category_one"
    android:summary="@string/prefs_summ_category_one" />

  <header
android:fragment="com.example.prefs.SettingsActivity$SettingsFragmentTwo"
    android:title="@string/prefs_category_two"
    android:summary="@string/prefs_summ_category_two" >
    <!-- key/value pairs can be included as arguments for the fragment. -->

    <extra

        android:name="someKey"

        android:value="someHeaderValue" />

    </header>
</preference-headers>
```

Với thuộc tính android:fragment, mỗi header khai báo một thể hiện của [PreferenceFragment](#) mà cần phải mở khi người dùng chọn header đó.

Phần tử <extras> cho phép ta truyền cặp **key-value** tới fragment trong một [Bundle](#). Fragment có thể thu lại đối số bằng cách gọi hàm [getArguments\(\)](#). Ta có thể truyền đối số tới fragment với các lý do khác nhau, nhưng lý do chính đáng là để tái sử dụng lớp con của lớp [PreferenceFragment](#) cho mỗi nhóm và dùng đối số để xác định tập tin preferences XML nào mà fragment cần tải.

**Ví dụ:** Đây là một fragment có thể được tái sử dụng cho nhiều nhóm thiết lập, khi mỗi header định nghĩa một đối số <extra> với khoá "settings":

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String settings = getArguments().getString("settings");
        if ("notifications".equals(settings)) {
            addPreferencesFromResource(R.xml.settings_wifi);
        }

        else if ("sync".equals(settings)) {
            addPreferencesFromResource(R.xml.settings_sync);
        }
    }
}
```

### Hiển thị header

Để hiển thị các preference header, ta phải thực thi phương thức `onBuildHeaders()` và gọi phương thức `loadHeadersFromResource()`.

### Ví dụ:

```
public class SettingsActivity extends PreferenceActivity {
    @Override
    public void onBuildHeaders(List<Header> target) {
        loadHeadersFromResource(R.xml.preference_headers, target);
    }
}
```

Khi người dùng chọn một mục từ danh sách các header, hệ thống mở một [PreferenceFragment](#) liên quan.

**Ghi chú:** Khi dùng preference header, lớp con của lớp [PreferenceActivity](#) không cần thực thi hàm `onCreate()`, bởi vì activity này chỉ dùng để tải header.

### Hỗ trợ preference headers cho phiên bản cũ

Nếu ứng dụng được sử dụng cho các phiên bản cũ hơn 3.0, ta vẫn có thể dùng header để cung cấp giao diện hai màn hình (two-pane layout) như khi chạy trên các phiên bản từ 3.0

về sau. Tất cả những gì cần thiết để tạo ra tập tin preferences XML với các phần tử [<Preference>](#) cơ bản giống như các mục header. Tuy nhiên, thay vì mở một [PreferenceScreen](#) mới, mỗi phần tử [<Preference>](#) gửi một [Intent](#) tới [PreferenceActivity](#) để xác định tập tin preference XML được tải.

### Ví dụ:

Ở đây là một tập tin XML cho preference headers được dùng trên Android 3.0 và mới hơn (res/xml/preference\_headers.xml):

```
<preference-headers xmlns:android="http://schemas.android.com/apk/res/android">
  <header
    android:fragment="com.example.prefs.SettingsFragmentOne"
    android:title="@string/prefs_category_one"
    android:summary="@string/prefs_summ_category_one" />
  <header
    android:fragment="com.example.prefs.SettingsFragmentTwo"
    android:title="@string/prefs_category_two"
    android:summary="@string/prefs_summ_category_two" />
</preference-headers>
```

Và sau đây là một tập tin preference cung cấp các header tương tự cho các phiên bản cũ hơn 3.0 (res/xml/preference\_headers\_legacy.xml):

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
  <Preference
    android:title="@string/prefs_category_one"
    android:summary="@string/prefs_summ_category_one" >
    <intent
      android:targetPackage="com.example.prefs"
      android:targetClass="com.example.prefs.SettingsActivity"
      android:action="com.example.prefs.PREFS_ONE" />
  </Preference>
  <Preference
    android:title="@string/prefs_category_two"
    android:summary="@string/prefs_summ_category_two" >
    <intent
      android:targetPackage="com.example.prefs"
      android:targetClass="com.example.prefs.SettingsActivity"
```

```
        android:action="com.example.prefs.PREFS_TWO" />
    </Preference>
</PreferenceScreen>
```

Vì để hỗ trợ cho <preference-headers> được thêm vào trong Android 3.0, hệ thống chỉ gọi hàm [onBuildHeaders\(\)](#) trong [PreferenceActivity](#) khi chạy trên Android 3.0 hay mới hơn. Để tải tập tin "legacy" headers (preference\_headers\_legacy.xml), ta phải kiểm tra phiên bản và nếu phiên bản cũ hơn 3.0 ([HONEYCOMB](#)), thì gọi hàm [addPreferencesFromResource\(\)](#) để tải tập tin legacy header.

### Ví dụ:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
        // Load the legacy preferences headers
        addPreferencesFromResource(R.xml.preference_headers_legacy);
    }
}

// Called only on Honeycomb and later
@Override
public void onBuildHeaders(List<Header> target) {
    loadHeadersFromResource(R.xml.preference_headers, target);
}
```

Việc duy nhất còn lại cần làm là điều khiển một [Intent](#) mà nó được đưa vào activity để xác định tập tin preference nào được tải. Vì vậy, cần lấy lại hoạt động của Intent và so sánh với chuỗi hoạt động đã biết, đã được dùng trong <intent> của tập tin preference XML:

```
final static String ACTION_PREFS_ONE = "com.example.prefs.PREFS_ONE";
...

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```

String action = getIntent().getAction();
if (action != null && action.equals(ACTION_PREFS_ONE)) {
    addPreferencesFromResource(R.xml.preferences);
}
...

else if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
    // Load the legacy preferences headers
    addPreferencesFromResource(R.xml.preference_headers_legacy);
}
}

```

Chú ý rằng các cuộc gọi liên tiếp tới [addPreferencesFromResource\(\)](#) sẽ lưu tất cả preference vào một danh sách, vì vậy phải bảo đảm rằng nó chỉ được gọi một lần bằng cách liên kết các điều kiện với lệnh else-if.

## 6. Đọc Preferences

Theo mặc định, tất cả preference của ứng dụng được lưu vào một tập tin mà có thể truy xuất bất kỳ từ nơi nào trong ứng dụng đó bằng cách dùng phương thức [PreferenceManager.getDefaultSharedPreferences\(\)](#). Phương thức này trả về đối tượng [SharedPreferences](#) chứa tất cả các cặp **key-value** liên kết với các đối tượng [Preference](#) được dùng trong [PreferenceActivity](#).

### Ví dụ:

Sau đây là cách đọc một giá trị của preference từ một activity bất kỳ trong ứng dụng:

```

SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
String syncConnPref =
sharedPref.getString(SettingsActivity.KEY_PREF_SYNC_CONN, "");

```

## 7. Lắng nghe những thay đổi của preference

Để thu được một thông báo khi có sự thay đổi của preference, giao diện [SharedPreferences.OnSharedPreferenceChangeListener](#) được thực hiện và phương thức [registerOnSharedPreferenceChangeListener\(\)](#) được gọi.



Giao diện này chỉ có một phương thức gọi lại (callback method) là `onSharedPreferenceChanged()`, và chúng ta có thể thực thi giao diện như là một phần của activity.

### Ví dụ:

```
public class SettingsActivity extends PreferenceActivity
    implements OnSharedPreferenceChangeListener {
    public static final String KEY_PREF_SYNC_CONN = "pref_syncConnectionType";
    ...

    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
        String key) {
        if (key.equals(KEY_PREF_SYNC_CONN)) {
            Preference connectionPref = findPreference(key);
            // Set summary to be the user-description for the selected value
            connectionPref.setSummary(sharedPreferences.getString(key, ""));
        }
    }
}
```

Trong ví dụ này, phương pháp kiểm tra sự thay đổi của các thiết lập là preference key đã biết. Phương thức `findPreference()` được gọi để lấy đối tượng `Preference` đã được thay đổi, vì vậy nó có thể sửa đổi bảng tóm tắt của một mục, là một mô tả về sự chọn lựa của người dùng. Khi thiết lập là một `ListPreference` hay một thiết lập nhiều chọn lựa khác nhau, ta nên gọi phương thức `setSummary()` khi thiết lập thay đổi để hiển thị trạng thái hiện hành (Chẳng hạn như thiết lập Sleep trong hình 5.5).

**Ghi chú:** Chúng ta nên cập nhật bảng tóm tắt cho `ListPreference` mỗi khi người dùng thay đổi preference để mô tả thiết lập hiện hành.

Cho việc quản lý vòng đời riêng trong activity, chúng ta nên đăng ký `SharedPreferences.OnSharedPreferenceChangeListener` trong phương thức `onResume()` và không đăng ký trong phương thức `onPause()`, xem đoạn mã lệnh sau:

```
@Override
protected void onResume() {
    super.onResume();
```

```

    getPreferenceScreen().getSharedPreferences()
        .registerOnSharedPreferenceChangeListener(this);
}

@Override
protected void onPause() {
    super.onPause();
    getPreferenceScreen().getSharedPreferences()
        .unregisterOnSharedPreferenceChangeListener(this);
}

```

**Chú ý:** Khi phương thức `registerOnSharedPreferenceChangeListener()` được gọi, bộ quản lý preference hiện thời không lưu trữ một tham chiếu đủ mạnh cho listener. Ta phải lưu trữ một tham chiếu đủ mạnh đến listener, nếu không nó sẽ là đối tượng dễ bị “thu gom rác thải”. Chúng ta nên giữ một tham chiếu tới listener trong các dữ liệu thể hiện của một đối tượng mà sẽ tồn tại miễn là có yêu cầu listener.

### Ví dụ:

Đoạn lệnh sau cho thấy kết quả là listener sẽ bị thu gom rác thải, và nó sẽ thất bại tại một thời gian không xác định trong tương lai.

```

prefs.registerOnSharedPreferenceChangeListener(
    // Bad! The listener is subject to garbage collection!
    new SharedPreferences.OnSharedPreferenceChangeListener() {
        public void onSharedPreferenceChanged(SharedPreferences prefs, String key) {
            // listener implementation
        }
    });

```

Ta nên viết lại như sau, để lưu trữ một tham chiếu đến listener trong một trường dữ liệu thể hiện của một đối tượng mà nó sẽ tồn tại khi listener là cần thiết.

```

SharedPreferences.OnSharedPreferenceChangeListener listener = new
SharedPreferences.OnSharedPreferenceChangeListener() { public void
onSharedPreferenceChanged(SharedPreferences prefs, String key) {

// listener implementation
}

```

```
};  
prefs.registerOnSharedPreferenceChangeListener(listener);
```

## 8. Quản lý việc sử dụng mạng

Từ Android 4.0, ứng dụng lưu trữ thiết lập của hệ thống cho phép người dùng thấy được có bao nhiêu dữ liệu mạng mà ứng dụng đang dùng trong tiến trình foreground và background. Từ đó, người dùng có thể tùy chỉnh mức độ truy cập dữ liệu của mình theo nhu cầu và mức phí.

Ví dụ, chúng ta có thể cho phép người dùng kiểm soát mức độ đồng bộ hoá dữ liệu trong ứng dụng, chọn chế độ chỉ cho phép ứng dụng thực hiện uploads/downloads qua Wi-Fi hay không, cho phép ứng dụng dùng dữ liệu khi chuyển vùng (roaming) hay không, ...

Khi ta muốn thêm vào các preference cần thiết trong [PreferenceActivity](#) để kiểm soát việc sử dụng dữ liệu của ứng dụng, ta nên thêm vào một intent filter với tham số [ACTION\\_MANAGE\\_NETWORK\\_USAGE](#) trong tập tin manifest.

### Ví dụ:

```
<activity android:name="SettingsActivity" ... >  
  <intent-filter>  
    <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />  
    <category android:name="android.intent.category.DEFAULT" />  
  </intent-filter>  
</activity>
```

Intent filter này báo cho hệ thống rằng đây là activity mà nó điều khiển việc sử dụng dữ liệu mạng của ứng dụng. Vì vậy, để người dùng kiểm tra lượng dữ liệu mà ứng dụng đang sử dụng từ các thiết lập của hệ thống, một nút nhấn cho phép xem các thiết lập của ứng dụng (*View application settings* button) được cung cấp để khởi chạy [PreferenceActivity](#) cho phép người dùng có thể điều chỉnh dung lượng dữ liệu mạng được sử dụng trong ứng dụng.

## 9. Xây dựng Preference tùy chọn

Android framework bao gồm một nhiều lớp con của [Preference](#), nó cho phép chúng ta xây dựng giao diện cho nhiều loại thiết lập khác nhau. Tuy nhiên, đôi khi chúng cần điều chỉnh một thiết lập mà chưa có giải pháp xây dựng sẵn (built-in solution), chẳng hạn như number picker hay date picker. Trong trường hợp này, chúng ta cần tạo ra một custom preference bằng kế thừa của lớp [Preference](#) hay một lớp con khác của nó.

Nếu dùng kế thừa của lớp **Preference** thì có một vài việc quan trọng cần làm là:

- Xác định giao diện người dùng mà nó xuất hiện khi người dùng chọn thiết lập.
- Lưu các giá trị của các thiết lập lúc thích hợp.
- Khởi chạy **Preference** với giá trị hiện hành (hay mặc định) khi nó kết hợp với đối tượng view.
- Cung cấp giá trị mặc định khi có yêu cầu của hệ thống.
- Nếu **Preference** cung cấp giao diện của riêng nó (chẳng hạn như một dialog), lưu và khôi phục trạng thái để điều khiển những thay đổi trong vòng đời (chẳng hạn khi người dùng quay màn hình).

Sau đây sẽ trình bày cách để thực hiện các việc trên:

### 9.1. Đặc tả giao diện người dùng

Nếu chúng ta kế thừa trực tiếp lớp **Preference**, ta cần phải thực thi phương thức **onClick()** để xử lý tương tác của người dùng. Tuy nhiên, hầu hết các thiết lập tùy chọn được trình bày bằng một dialog, là kế thừa của lớp **DialogPreference**. Khi kế thừa lớp **DialogPreference**, ta phải gọi hàm **setDialogLayoutResources()** trong phương thức xây dựng của lớp để xác định layout cho dialog.

#### Ví dụ:

Đoạn mã sau đây là hàm tạo cho một **DialogPreference** tùy chọn mà nó khai báo layout và xác định văn bản cho các nút nhấn dialog tích cực và dialog thụ động:

```
public class NumberPickerPreference extends DialogPreference {
    public NumberPickerPreference(Context context, AttributeSet attrs) {
        super(context, attrs);

        setDialogLayoutResource(R.layout.numberpicker_dialog);
        setPositiveButton(android.R.string.ok);
        setNegativeButton(android.R.string.cancel);

        setDialogIcon(null);
    }
    ...
}
```

### 9.2. Lưu giá trị của các thiết lập

Chúng ta có thể lưu một giá trị cho thiết lập bất kỳ lúc nào bằng cách gọi một phương thức `persist*()` của lớp `Preference`, chẳng hạn như `persistInt()` nếu giá trị của thiết lập là một số nguyên hay `persistBoolean()` để lưu một giá trị boolean.

**Chú ý:** Mỗi `Preference` cần lưu chỉ dùng một loại dữ liệu, vì vậy ta phải dùng phương thức `persist*()` thích hợp cho loại dữ liệu được dùng bởi `Preference` tùy chọn.

Việc chọn phương thức `persist*()` phụ thuộc vào lớp `Preference` nào được chọn để kế thừa. Nếu kế thừa lớp `DialogPreference`, thì ta chỉ gọi phương thức `persist*()` khi dialog đóng để có kết quả tích cực (Người dùng chọn nút "OK").

Khi một `DialogPreference` đóng, hệ thống gọi phương thức `onDialogClosed()`. Phương thức này bao gồm một đối số mà nó xác định kết quả là "positive", nếu giá trị là true, thì người dùng đã chọn nút tích cực và ta cần lưu lại giá trị mới.

Ví dụ:

```
@Override
protected void onDialogClosed(boolean positiveResult) {
    // When the user selects "OK", persist the new value
    if (positiveResult) {
        persistInt(mNewValue);
    }
}
```

Trong ví dụ này, `mNewValue` là một thành viên của lớp, nó giữ giá trị hiện hành của thiết lập. Giá trị này được lưu vào tập tin `SharedPreferences` khi gọi `persistInt()` (sử dụng một cách tự động **key** mà nó được xác định trong tập tin XML cho `Preference` này).

### 9.3. Khởi tạo giá trị hiện tại

Khi hệ thống thêm `Preference` vào màn hình, nó gọi phương thức `onSetInitialValue()` để thông báo thiết lập có một giá trị đã tồn tại hay không. Nếu không có một giá trị đã tồn tại, việc gọi phương thức này sẽ cung cấp một giá trị mặc định.

Phương thức `onSetInitialValue()` thông qua một giá trị boolean, `restorePersistedValue`, để chỉ rằng có hay không một giá trị đã được lưu sẵn cho thiết lập. Nếu nó là **true**, thì ta sẽ thu về giá trị đã tồn tại bằng việc gọi một trong những phương thức `getPersisted*()` của lớp `Preference`, chẳng hạn như phương thức `getPersistedInt()` cho một giá trị nguyên. Ta

thường muốn thu về một giá trị đã tồn tại, vì vậy ta có thể cập nhật một cách chính xác giao diện người dùng phản ánh giá trị đã lưu trước đó.

Nếu `restorePersistedValue` là **false**, thì ta nên dùng giá trị mặc định được thông qua trong đối số thứ hai.

```
@Override
protected void onSetInitialValue(boolean restorePersistedValue, Object defaultValue) {
    if (restorePersistedValue) {
        // Restore existing state
        mCurrentValue = this.getPersistedInt(DEFAULT_VALUE);
    }

    else {
        // Set default state from the XML attribute
        mCurrentValue = (Integer) defaultValue;
        persistInt(mCurrentValue);
    }
}
```

Mỗi phương thức `getPersisted*()` lấy một đối số mà nó xác định giá trị mặc định để dùng trong trường hợp không có giá trị đã tồn tại trước đó hoặc **key** không tồn tại. Trong ví dụ trên, một hằng cục bộ được dùng để xác định giá trị mặc định trong trường hợp phương thức `getPersistedInt()` không thể trả về một giá trị đã tồn tại.

**Chú ý:** ta không thể dùng `defaultValue` như một giá trị mặc định trong phương thức `getPersisted*()`, bởi vì giá trị của nó luôn luôn là **null** khi `restorePersistedValue` là **true**.

#### 9.4. Cung cấp một giá trị mặc định

Nếu thể hiện của lớp `Preference` xác định một giá trị mặc định (với thuộc tính `android:defaultValue`), thì hệ thống gọi phương thức `onGetDefaultValue()` khi nó khởi tạo đối tượng để lấy giá trị. Ta phải thực thi phương thức này để cho hệ thống lưu giá trị mặc định trong `SharedPreferences`.

**Ví dụ:**

```
@Override
protected Object onGetDefaultValue(TypedArray a, int index) {
```

```
return a.getInteger(index, DEFAULT_VALUE);  
}
```

Các đối số của phương thức cung cấp mọi thứ mà ta cần: mảng các thuộc tính và vị trí chỉ số của android: defaultValue, mà ta phải lấy. Lý do mà ta phải thực hiện phương thức này để trích xuất các giá trị mặc định của thuộc tính là bởi vì ta phải chỉ định một giá trị cục bộ mặc định cho các thuộc tính trong trường hợp giá trị là không xác định.

### 9.5. Lưu và khôi phục trạng thái của Preference

Giống như một [View](#) trong một layout, lớp con của [Preference](#) có thể thực hiện sự lưu trữ và khôi phục trạng thái của nó trong trường hợp activity hay fragment được khởi động lại (chẳng hạn như khi người dùng quay màn hình). Để lưu và khôi phục trạng thái của lớp [Preference](#), ta phải thực thi các phương thức [onSaveInstanceState\(\)](#) và [onRestoreInstanceState\(\)](#).

Trạng thái của [Preference](#) được định nghĩa bởi một đối tượng mà nó thực thi giao diện [Parcelable](#). Android framework cung cấp một đối tượng như là điểm khởi đầu để định nghĩa một đối tượng trạng thái: lớp [Preference.BaseSavedState](#).

Để xác định cách mà lớp [Preference](#) lưu trạng thái của nó, ta cần kế thừa lớp [Preference.BaseSavedState](#). Ta cần phải ghi đè một vài phương thức và định nghĩa đối tượng [CREATOR](#).

Với hầu hết các ứng dụng, ta có thể dùng các lệnh sau và thay đổi các dòng lệnh mà nó điều khiển giá trị nếu lớp con của [Preference](#) lưu một loại dữ liệu khác một số nguyên.

```
private static class SavedState extends BaseSavedState {  
    // Member that holds the setting's value  
    // Change this data type to match the type saved by your Preference  
    int value;  
  
    public SavedState(Parcelable superState) {  
        super(superState);  
    }  
  
    public SavedState(Parcel source) {  
        super(source);  
        // Get the current preference's value
```

```

        value = source.readInt(); // Change this to read the appropriate data type
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        super.writeToParcel(dest, flags);
        // Write the preference's value
        dest.writeInt(value); // Change this to write the appropriate data type
    }

    // Standard creator object using an instance of this class
    public static final Parcelable.Creator<SavedState> CREATOR =
        new Parcelable.Creator<SavedState>() {

        public SavedState createFromParcel(Parcel in) {
            return new SavedState(in);
        }

        public SavedState[] newArray(int size) {
            return new SavedState[size];
        }
    };
}

```

Với sự thực thi [Preference.BaseSavedState](#) ở trên đã thêm vào ứng dụng (thường như một lớp con của lớp con [Preference](#)), sau đó ta cần thực thi các phương thức [onSaveInstanceState\(\)](#) và [onRestoreInstanceState\(\)](#) cho lớp con [Preference](#).

Ví dụ:

```

@Override
protected Parcelable onSaveInstanceState() {
    final Parcelable superState = super.onSaveInstanceState();
    // Check whether this Preference is persistent (continually saved)
    if (isPersistent()) {
        // No need to save instance state since it's persistent,
        // use superclass state
        return superState;
    }
}

```



```

// Create instance of custom BaseSavedState
final SavedState myState = new SavedState(superState);
// Set the state's value with the class member that holds current
// setting value
myState.value = mNewValue;
return myState;
}

@Override
protected void onRestoreInstanceState(Parcelable state) {
    // Check whether we saved the state in onSaveInstanceState
    if (state == null || !state.getClass().equals(SavedState.class)) {
        // Didn't save the state, so call superclass
        super.onRestoreInstanceState(state);
        return;
    }

    // Cast state to custom BaseSavedState and pass to superclass
    SavedState myState = (SavedState) state;
    super.onRestoreInstanceState(myState.getSuperState());

    // Set this Preference's widget to reflect the restored state
    mNumberPicker.setValue(myState.value);
}

```

## 10. Các ví dụ

### Ví dụ 1: Lưu thiết lập của người dùng với đối tượng **SharedPreferences**

Như trên đã trình bày, Android cung cấp đối tượng **SharedPreferences** nhằm giúp cho người dùng lưu các thiết lập của ứng dụng một đơn giản.

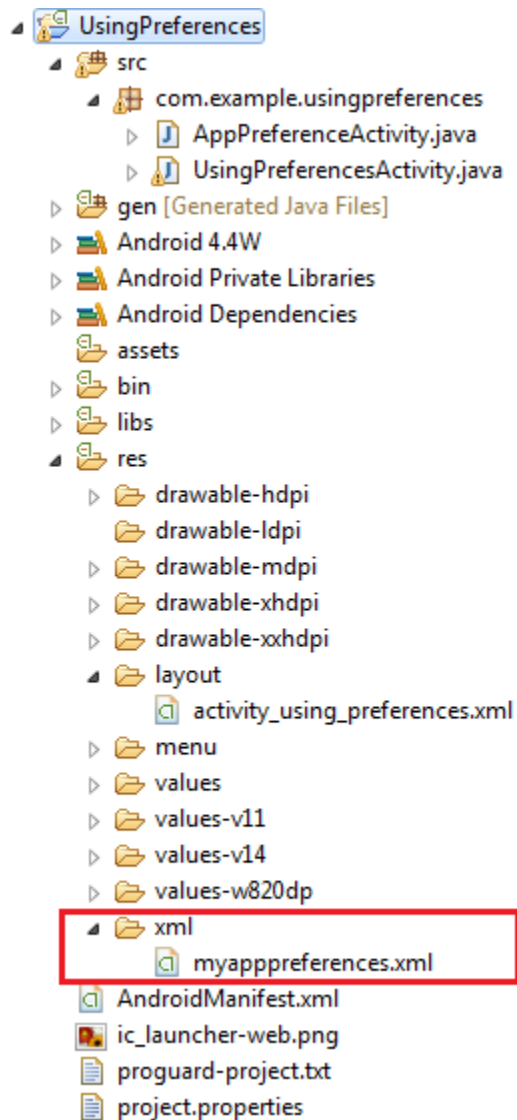
Ví dụ, ứng dụng của bạn có một tùy chọn cho phép người dùng xác định độ lớn phông chữ của văn bản được hiển thị trong một ứng dụng. Khi đó, ứng dụng này cần phải nhớ độ lớn được thiết lập bởi người dùng để lần sau người đó sử dụng ứng dụng một lần nữa, nó có thể lấy lại thiết lập độ lớn đã được lưu. Để thực hiện điều này, ta có nhiều lựa chọn. Chúng ta có thể lưu dữ liệu vào một tập tin, nhưng ta phải thực hiện một số thao tác quản

lý tập tin quen thuộc, chẳng hạn như ghi dữ liệu vào tập tin, xác định số ký tự đọc từ tập tin, .... Ngoài ra, nếu ta có một nhiều thông tin cần lưu, chẳng hạn như kích cỡ chữ, kiểu chữ, màu nền, ... thì việc ghi vào tập tin sẽ trở nên khó khăn hơn.

Một cách khác để ghi một tập tin văn bản là sử dụng cơ sở dữ liệu, nhưng việc sử dụng cơ sở dữ liệu là không cần thiết đối với một dữ liệu đơn giản. Thay vào đó, ta có thể sử dụng đối tượng **SharedPreferences**. SharedPreferences lưu dữ liệu thông qua việc sử dụng các cặp **tên / giá trị** - chỉ định tên cho các dữ liệu muốn lưu, sau đó tên và giá trị của nó sẽ được tự động lưu vào một tập tin XML.

Các bước thực hiện như sau (các tên project, class, tập tin, thư mục sau đây chỉ là ví dụ):

- (1) Tạo ra một Android project, được đặt tên là **UsingPreferences**.
- (2) Tạo một thư mục con, trong thư mục **res**, thư mục con này được đặt tên là **xml** (nhấp chuột phải vào thư mục res, chọn New, chọn Folder, đặt tên cho thư mục mới, chọn Finish). Trong thư mục **xml** tạo ra một tập tin XML (nhấp chuột phải vào thư mục xml, chọn New, chọn File, đặt tên cho tập tin mới, chọn Finish), được đặt tên là **myapppreferences.xml** (xem hình 5.6).



**Hình 5.6:** Thư mục của project UsingPreferences

(3) Hoàn thành nội dung của tập tin **myapppreferences.xml** như sau:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory android:title="Category 1">
    <CheckBoxPreference
      android:title="Checkbox"
      android:defaultValue="false"
      android:summary="True or False"
      android:key="checkboxPref" />
  </PreferenceCategory>
</PreferenceScreen>
```

```

</PreferenceCategory>
<PreferenceCategory android:title="Category 2">
    <EditTextPreference
        android:summary="Enter a string"
        android:defaultValue="[Enter a string here]"
        android:title="Edit Text"
        android:key="editTextPref" />
    <RingtonePreference
        android:summary="Select a ringtone"
        android:title="Ringtones"
        android:key="ringtonePref" />
    <PreferenceScreen
        android:title="Second Preference Screen"
        android:summary="Click here to go to the second Preference
Screen"
        android:key="secondPrefScreenPref" >
        <EditTextPreference
            android:summary="Enter a string"
            android:title="Edit Text (second Screen)"
            android:key="secondEditTextPref" />
        </PreferenceScreen>
    </PreferenceCategory>
</PreferenceScreen>

```

- (4) Xây dựng một lớp mới có tên là **AppPreferenceActivity** dưới tên miền của ứng dụng (nhấp chuột phải và tên miền của ứng dụng trong thư mục src, chọn New, chọn Class, nhập vào tên lớp trong trường Name, chọn Finish).
- (5) Hoàn thành nội dung của lớp **AppPreferenceActivity** như sau:

```

package com.example.usingpreferences;
import android.os.Bundle;
import android.preference.PreferenceActivity;
public class AppPreferenceActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.myapppreferences);
    }
}

```

- (6) Thêm vào tập tin AndroidManifest.xml một mục mới cho lớp **AppPreferenceActivity**, như sau (dòng tên đậm):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.usingpreferences"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="20" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".UsingPreferencesActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AppPreferenceActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action
                    android:name="com.example.AppPreferenceActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- (7) Mở tập tin layout **activity\_using\_preferences.xml**, bỏ TextView và thêm vào các dòng lệnh (được in đậm) như sau:

```
<LinearLayout 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"
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:orientation="vertical"
tools:context="com.example.usingpreferences.UsingPreferencesActivity" >
```

```
<Button
android:id="@+id/btnPreferences"
android:text="Load Preferences Screen"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:onClick="onClickLoad"/>>
<Button
android:id="@+id/btnDisplayValues"
android:text="Display Preferences Values"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:onClick="onClickDisplay"/>>
<EditText
android:id="@+id/txtString"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
<Button
android:id="@+id/btnModifyValues"
android:text="Modify Preferences Values"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:onClick="onClickModify"/>>

</LinearLayout>
```

- (8) Thêm các lệnh cần thiết vào lớp activity chính, đó là **UsingPreferencesActivity**, như sau (các lệnh được in đậm):

```
package com.example.usingpreferences;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;

public class UsingPreferencesActivity extends Activity {

    @Override
```

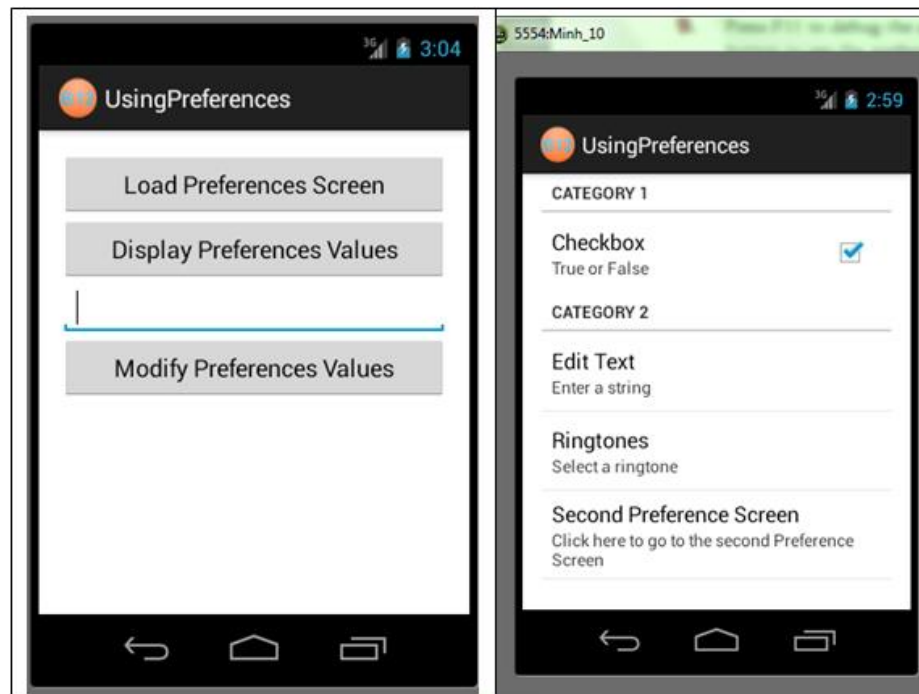
```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_using_preferences);
}

public void onClickLoad(View view) {
    Intent i = new
    Intent("com.example.AppPreferenceActivity");
    startActivity(i);
}
}

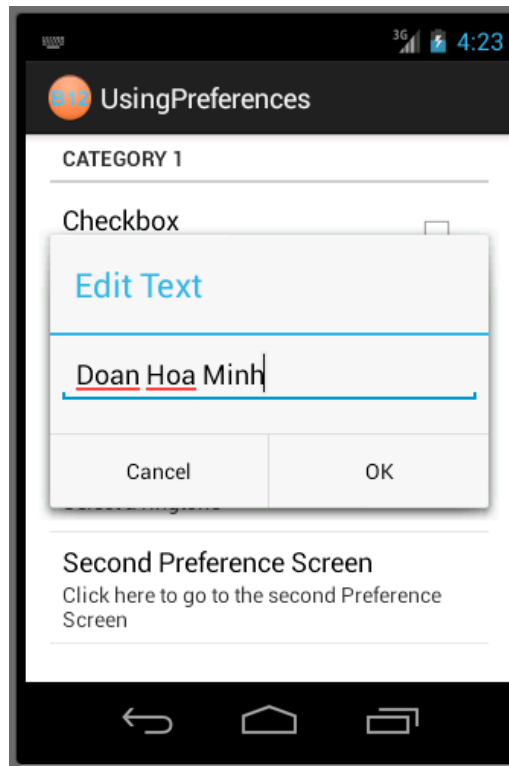
```

- (9) Nhấn F11 để debug ứng dụng với Android emulator. Nhấp Load Preferences Screen để hiển thị màn hình Preferences (hình 5.7).



**Hình 5.7:** Giao diện chính (trái) và giao diện Preferences (phải)

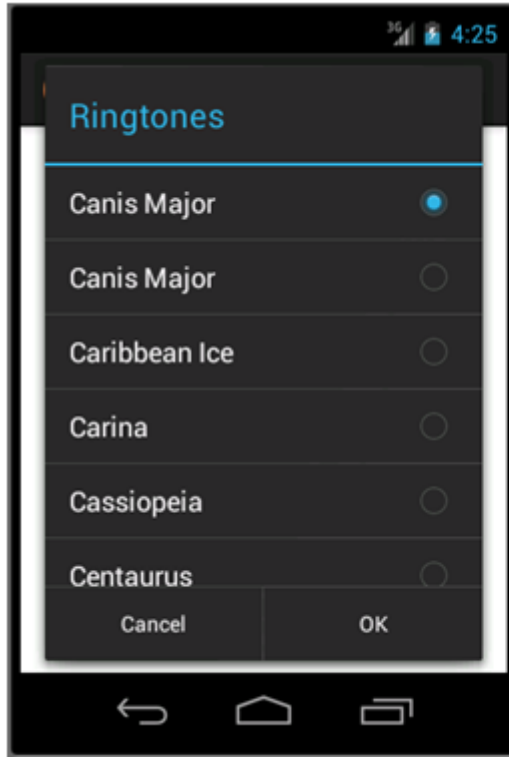
- (10) Nhấp checkbox để chọn 1 trong 2 giá trị checked hoặc unchecked. Lưu ý 2 loại Category 1 và Category 2. Nhập vào EditText 1 text (hình 5.8) và nhấp OK.



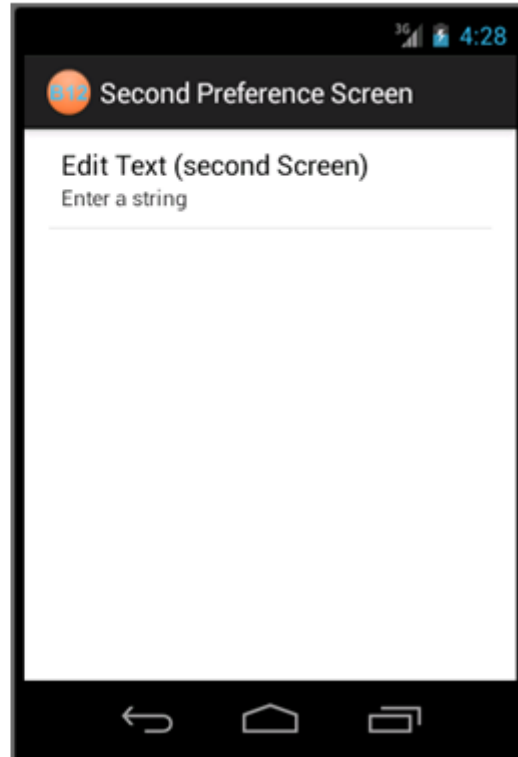
**Hình 5.8:** EditText

- (11) Nhấp vào mục **Ringtones** để chọn 1 trong 2 chế độ **default ringtone** hoặc **silent** (hình 5.9). Nếu chúng ta thể ứng dụng trên thiết bị thật, chúng ta có thể có nhiều lựa chọn hơn từ danh sách ringtones.
- (12) Nhấp vào mục **Second Preference Screen** ứng dụng sẽ chuyển tới màn hình kế tiếp (hình 5.10).





**Hình 5.9:** Chọn ringtones




**Hình 5.10:** Màn hình thứ 2

- (13) Để trở về màn hình trước, ta nhấp vào nút Back. Để thoát khỏi màn hình preferences ta cũng tiếp tục nhấp Back.
- (14) Khi ta có thay đổi các sở thích của dữ liệu người dùng, một tập tin được sinh ra trong thư mục:

**`/data/data/com.example.UsingPreferences/shared_prefs`**

của thẻ nhớ của Android emulator. Để kiểm tra điều này, hãy vào DDMS perspective trong Eclipse và quan sát File Explorer tab, ta sẽ thấy một tập tin XML có tên là `com.example.UsingPreferences_preferences.xml`.

- (15) Nếu ta mở tập tin này (*Mở DDMS, chọn File Explorer, chọn file muốn mở, chọn Pull a file from the device tại biểu tượng  ở góc phải của màn hình, chọn lưu tập tin này vào một thư mục nào đó trên PC, mở file này bằng Microsoft Word hoặc Notepad*) và kiểm tra nội dung của nó, ta sẽ thấy như sau:

và kiểm tra nội dung của nó, ta sẽ thấy như sau:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
```

```
<string name="editTextPref">[Enter a string here]</string>
<string name="ringtonePref"></string>
</map>
```

## Ví dụ 2: Lấy và sửa đổi tùy chỉnh của người dùng

Chúng ta sẽ lấy ví dụ 1 và chỉnh sửa cho phần này.

- (1) Chỉnh sửa file **UsingPreferencesActivity.java** ở ví dụ 1 (thêm vào các dòng in đậm):

```
Package com.example.usingpreferences;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class UsingPreferencesActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_using_preferences);
    }

    public void onClickLoad(View view) {
        Intent i = new Intent("com.example.AppPreferenceActivity");
        startActivity(i);
    }

    public void onClickDisplay(View view) {
        SharedPreferences appPrefs =
            getSharedPreferences("com.example.usingpreferences",
                MODE_PRIVATE);
        DisplayText(appPrefs.getString("editTextPref", ""));
    }
    public void onClickModify(View view) {
        SharedPreferences appPrefs =
```

```

getSharedPreferences("com.example.usingpreferences",
    MODE_PRIVATE);
SharedPreferences.Editor prefsEditor = appPrefs.edit();
prefsEditor.putString("editTextPref",
    ((EditText) findViewById(R.id.txtString)).getText().toString());
prefsEditor.commit();
}
private void DisplayText(String str) {
    Toast.makeText(getApplicationContext(), str, Toast.LENGTH_LONG).show();
}

```

```

}

```

### Giải thích:

Trong phương thức **onClickDisplay()**, trước tiên chúng ta dùng phương thức **getSharedPreferences()** để thu một **thể hiện** của **lớp SharedPreferences**. Chúng ta làm như vậy để xác định tên của tập tin XML (trong trường hợp này là “com.example.usingpreferences\_preferences,” sử dụng định dạng: **<PackageName>\_preferences**). Để thu được một chuỗi tùy chọn (preference), ta dùng phương thức **getString()**, thông qua phương thức này, ta thu được tùy chọn mà mong muốn:

```

public void onClickDisplay(View view) {

    SharedPreferences appPrefs =
getSharedPreferences("com.example.UsingPreferences_preferences",
    MODE_PRIVATE);

    DisplayText(appPrefs.getString("editTextPref", ""));

}

```

Hằng **MODE\_PRIVATE** chỉ rằng tập tin preference tập có thể được mở bởi ứng dụng đã sinh ra nó.

Trong phương thức **onClickModify()**, chúng ta đã tạo ra đối tượng **SharedPreferences.Editor** bởi phương thức **edit()** của lớp **SharedPreferences**. Để thay đổi giá trị của chuỗi tùy chọn, ta dùng phương thức **putString()** **method**. Để lưu những thay đổi trong tập tin preferences, ta dùng phương thức **commit()**:

```

public void onClickModify(View view) {

    SharedPreferences appPrefs =
    getSharedPreferences("com.example.UsingPreferences_preferences",
    MODE_PRIVATE);

    SharedPreferences.Editor prefsEditor = appPrefs.edit();

    prefsEditor.putString("editTextPref",((EditText)
    findViewById(R.id.txtString)).getText().toString());

    prefsEditor.commit();

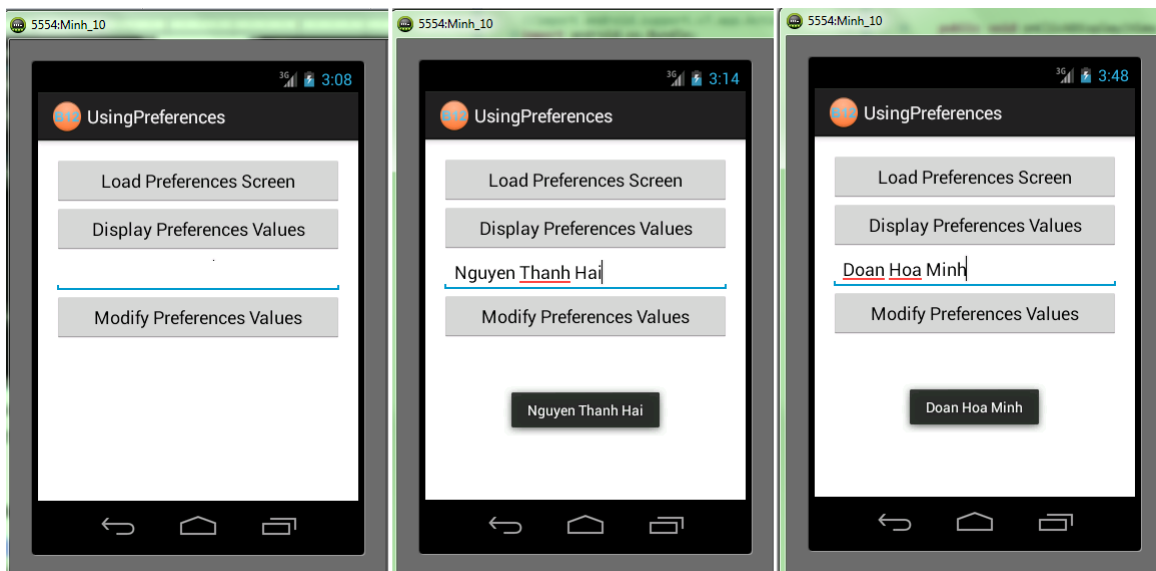
}

```

(2) Nhấn F11 để chạy lại ứng dụng. Sau đó nhấn nút “Display Preferences Values”, giá trị của đối tượng EditText là trống (Hình 5.11.a).

Nhập vào ô EditText một văn bản, ví dụ là “Nguyen Thanh Hai” và nhấn “Modify Preferences Values” giá trị mới của EditText được lưu. Nhấn “Display Preferences Values” lần nữa, ta thấy chuỗi ký tự “Nguyen Thanh Hai” hiển thị (Hình 5.11.b)

Nhập lại giá trị mới cho EditText, ví dụ là “Doan Hoa Minh” và nhấn “Modify Preferences Values” giá trị mới của EditText được lưu. Nhấn “Display Preferences Values” lần nữa, ta thấy chuỗi ký tự “Doan Hoa Minh” hiển thị (Hình 5.11.c).



(a) (b)

(c)

**Hình 5.11:** Giao diện của UsingPreferencesActivity

### (3) Sửa tên mặc định của file tùy chỉnh

Tên mặc định của file tùy chỉnh (Preferences) trong ví dụ này là **com.example.usingpreferences\_preferences.xml**, trong đó tên package như phần tiền tố. Tuy nhiên chúng ta cũng có thể tùy chỉnh lại tên tập tin này. Hãy thực hiện các bước sau:

- Thêm những dòng code in đậm vào AppPreferenceActivity.java:

```
package com.example.usingpreferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;
import android.preference.PreferenceManager;
public class AppPreferenceActivity extends PreferenceActivity {
    @SuppressWarnings("deprecation")
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PreferenceManager prefMgr = getPreferenceManager();
        prefMgr.setSharedPreferencesName("appPreferences");

        //---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.myapppreferences);
    }
}
```

Trong ví dụ, ta dùng lớp **PreferenceManager** để đổi tên của tập tin shared preferences thành **appPreferences.xml**.

Để thực hiện điều này, ta điều chỉnh tập tin **UsingPreferencesActivity.java** như sau:

- (1) Trong phương thức void onClickDisplay ta thay dòng lệnh

```
SharedPreferences appPrefs =
getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
MODE_PRIVATE);
```

bằng dòng lệnh

```
SharedPreferences appPrefs =
getSharedPreferences("appPreferences", MODE_PRIVATE);
```

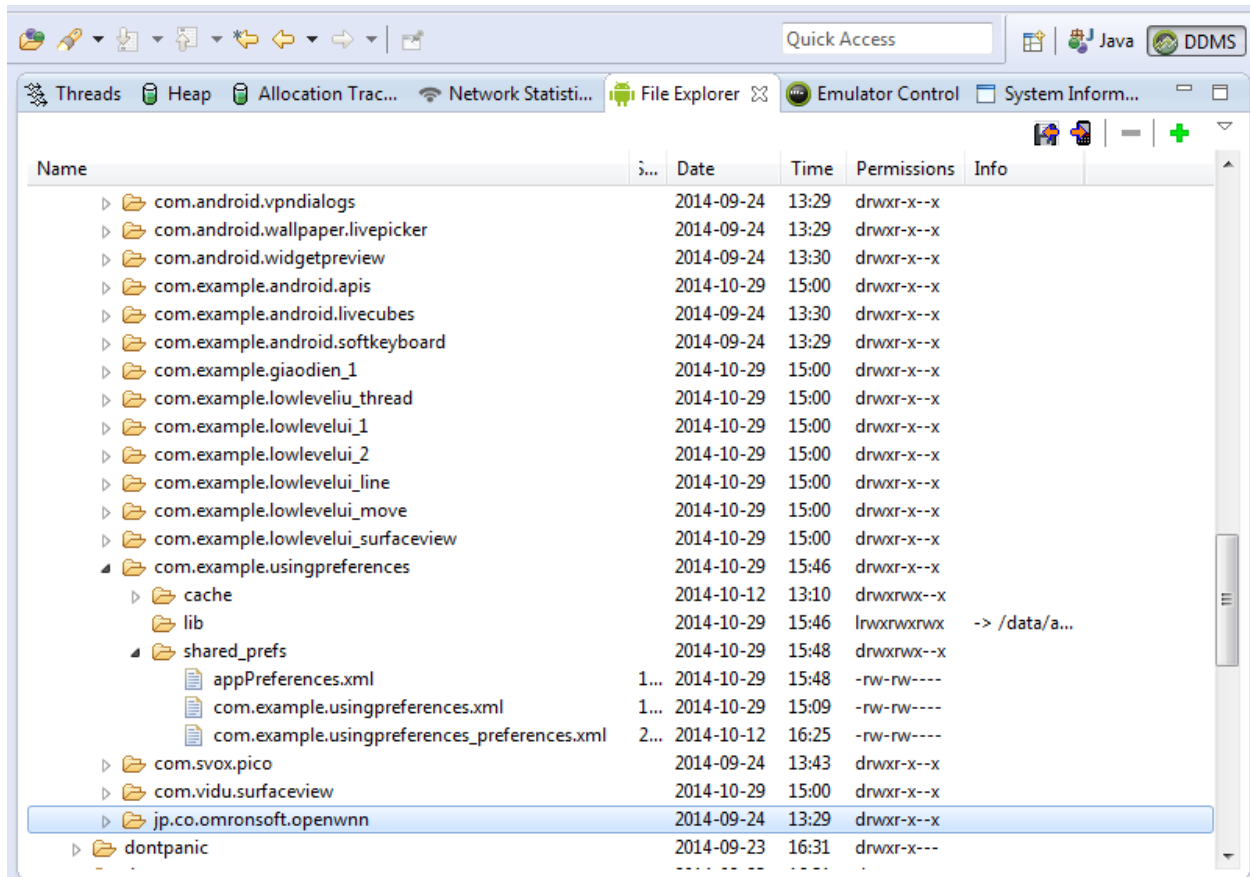
- (2) Trong phương thức void onClickModify ta thay dòng lệnh

```
SharedPreferences appPrefs =
getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
MODE_PRIVATE);
```

bằng dòng lệnh

```
SharedPreferences appPrefs =  
getSharedPreferences("appPreferences", MODE_PRIVATE);
```

(3) Chạy lại ứng dụng, mở **DDMS**, chọn mục **File Explorer**, ta sẽ thấy sẽ thấy tên tập tin là **appPreferences.xml** thay vì **myappPreferences.xml** như đã đặt trước đó (Hình 5.12).



Hình 5.12: Màn hình DDMS

### Tóm lại:

Để lưu dữ liệu hay các chọn lựa của người dùng trong một ứng dụng với đối tượng **SharedPreferences** ta thực hiện các bước cơ bản sau:

- Mở một thư mục trong thư mục **res** của ứng dụng và tạo ra tập tin **xml** để lưu dữ liệu người dùng trong đó.
- Tạo ra lớp con của lớp **PreferenceActivity** và sử dụng các phương thức của lớp này, Phương thức **getSharedPreferences**, hàm này trả về đối tượng **SharedPreferences** và nó có 2 đối:

```
SharedPreferences pre=getSharedPreferences("my_data", MODE_PRIVATE);
```

Đối số 1 là tên tập tin để lưu thiết lập (setting), đối số 2 là kiểu lưu. Chú ý là đối số 1 ta chỉ ghi tên tập tin (không cần ghi phần mở rộng, vì phần mở rộng mặc nhiên của nó là .xml khi ta lưu thành công), đối số 2 thường ta để là MODE\_PRIVATE.

- **Tạo đối tượng Editor** để cho phép chỉnh sửa dữ liệu:

```
SharedPreferences.Editor edit=pre.edit();
```

- Đưa dữ liệu muốn lưu trữ vào đối tượng edit bằng các phương thức:

```
edit.putXXX("key","value");
```

Tùy vào kiểu dữ liệu ta muốn lưu trữ mà XXX được thay thế bởi các kiểu dữ liệu phù hợp:

```
editor.putString("user", "dhminh");
```

```
editor.putString("pwd", "vidu01");
```

```
editor.putBoolean("checked", true);
```

- Lưu trạng thái bằng cách gọi dòng lệnh: **editor.commit();**

Sau khi 4 bước trên được thực hiện thì chương trình sẽ lưu được thiết lập, mặc định phần mở rộng là .xml (tức là thiết lập được lưu dưới định dạng tập tin XML). Ở đây, ta đặt tên là my\_data có nghĩa là chương trình sẽ tạo ra tập tin **my\_data.xml** (có thể mở DDMS để xem), ADT sẽ lưu tập tin này vào thư mục **shared\_prefs**.