

5.3. Интерфейс настройки приложения

Сайт: [Samsung Innovation Campus](https://innovationcampus.ru)

Курс: Мобильная разработка на Kotlin

Книга: 5.3. Интерфейс настройки приложения

Напечатано:: Murad Rezvan

Дата: понедельник, 3 июня 2024, 17:59

Оглавление

- 1. 5.3.1. Настройки приложения. AndroidX Preference library
- 2. 5.3.2. Интерфейс экрана настроек
- 3. 5.3.3. Отображение настроек
- 4. 5.3.4. Пример обработки изменения настроек приложения

1. 5.3.1. Настройки приложения. AndroidX Preference library

Во многих приложениях можно увидеть настройки. Они позволяют пользователям адаптировать приложение под себя, настроить внешний вид и режим работы. Кроме того, в экранах настройки зачастую предоставлены ссылки на внешнюю информацию (политика конфиденциальности, лицензии с открытым исходным кодом и многое другое). Данные экраны можно разрабатывать самостоятельно, при этом требуется не только создать пользовательский интерфейс экрана настроек, но и прописать необходимую логику поведения приложения при их изменении.

Ранее мы уже рассматривали [Android Preferences](#), и показали как их задействовать для хранения своих данных. Теперь посмотрим, как они используются для хранения настроек приложения.

Android Jetpack содержит библиотеку настроек [Preference Library](#), которая в значительной степени помогает разработчикам в создании экранов настроек. С помощью библиотеки AndroidX Preference library можно создавать интерактивные экраны настроек без необходимости взаимодействия с хранилищем устройства или управления пользовательским интерфейсом. Для использования данной библиотеки в файл build.gradle вашего проекта необходимо добавить следующие зависимости:

```
dependencies {  
    def preference_version = "1.1.1"  
    implementation "androidx.preference:preference-ktx:$preference_version"  
}
```

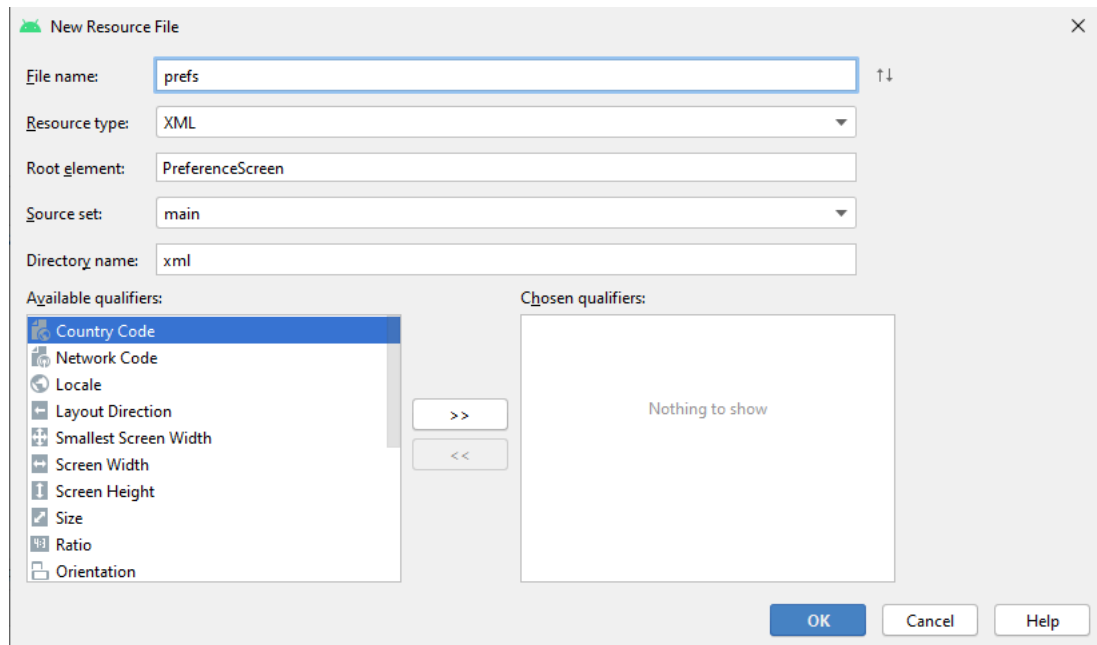
В данном разделе рассмотрим ключевые возможности библиотеки, посмотрим как создавать интерфейс настройки приложения и обрабатывать изменения настроек.

2. 5.3.2. Интерфейс экрана настроек

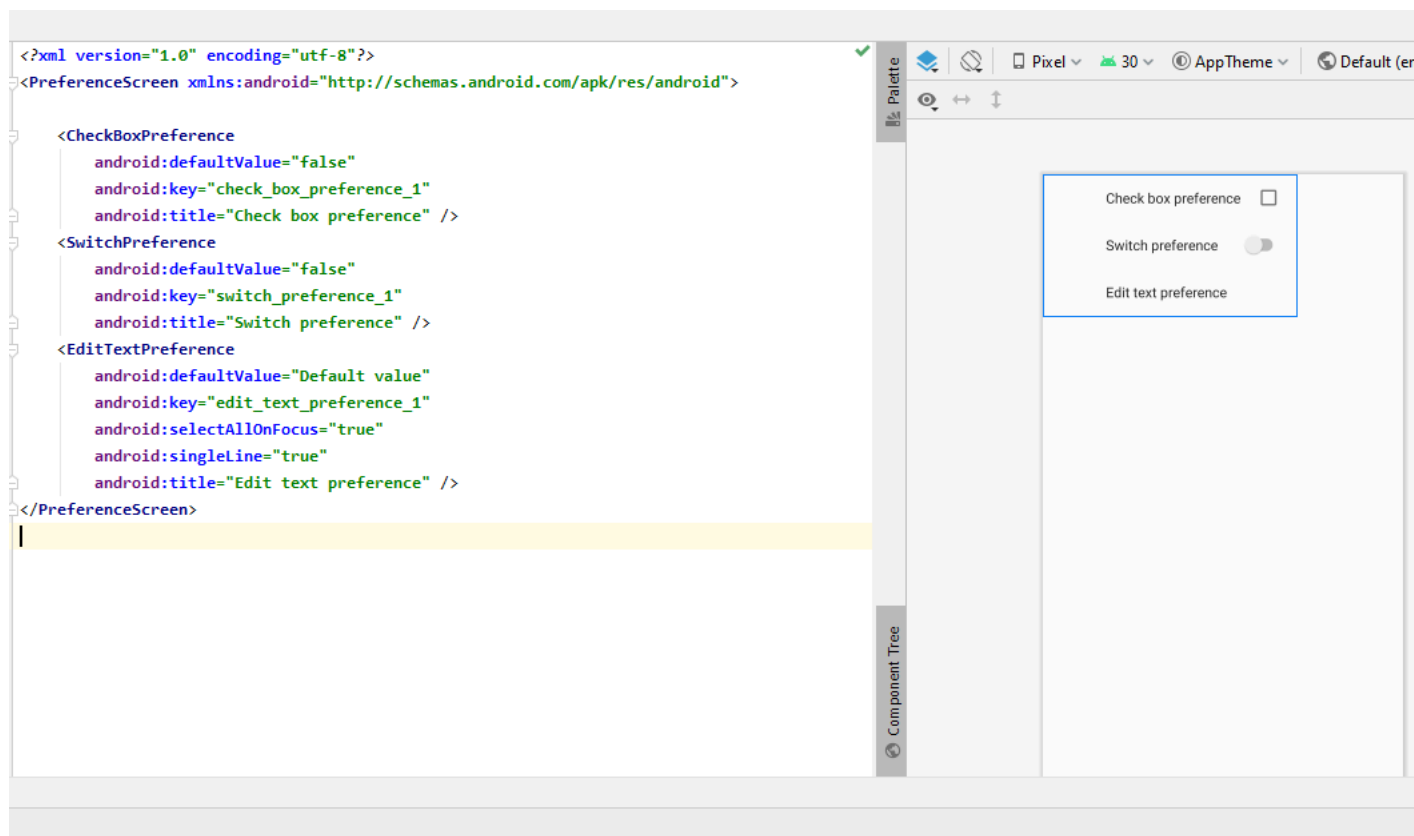
Прежде чем создавать экран настроек приложения, необходимо рассмотреть иерархию настроек. Данная иерархия используется для определения различных свойств настроек, которые могут быть установлены в приложении. Существует два способа определения иерархии настроек: программно или в файле XML. Способ отображения наших настроек в соответствующий компонент приложения (активность или фрагмент), будет зависеть от того, как мы изначально определяем иерархию настроек.

В случае если для построения иерархии настроек мы используем XML, необходимо определить новый файл ресурсов PreferenceScreen в каталоге res/xml. Далее можно определить различные настройки, которые мы хотим отображать на этом экране.

Например можно создать xml-файл с настройками res/xml/pref.xml образом:



После этого создастся xml-файл с корневым элементом PreferenceScreen, куда можно добавлять остальные xml-элементы настроек (добавлять можно вручную или с помощью конструктора). Далее необходимо определить различные настройки, которые мы хотим отображать на экране настроек. Если мы создаем настройки программно, то требуется перейти непосредственно к компоненту, который мы собираемся использовать для нашего экрана.



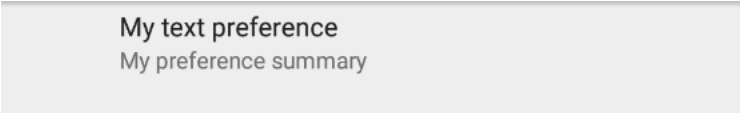
Существует множество различных типов настроек, которые можно использовать, но мы начнем с самого простого из них - текстового элемента.

```
<Preference
    android:key="preference_key"
    android:title="@string/preference_title"
    android:summary="@string/preference_summary" />
```

В данном случае определено три атрибута элемента Preference:

- **key** - ключ используется для сохранения и извлечения значения настроек;
- **title** - заголовок, используемый в отображении настроек;
- **summary** - описание настроек.

В итоге на экране будет показано следующее:

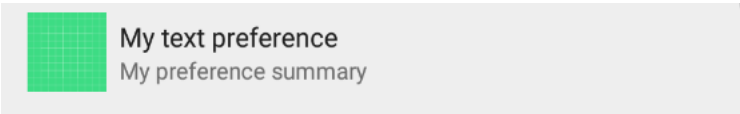


Если мы создаем настройки программно, необходимо создать новый экземпляр класса Preference:

```
val simplePreference = Preference(context).apply {
    key = "my_preference"
    title = "My title"
    summary = "My summary"
}
```

Также можно устанавливать иконку для настройки, используя атрибут **android:icon**:

```
<Preference
    android:key="preference_key"
    android:title="@string/preference_title"
    android:summary="@string/preference_summary"
    android:icon="@drawable/ic_launcher_background" />
```

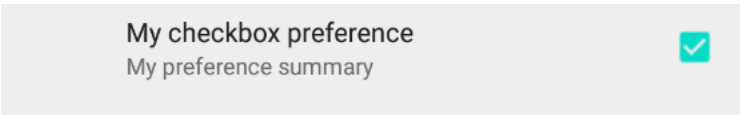


Если мы создаем настройки программно, установим атрибут **icon** экземпляру класса Preference:

```
val iconPreference = Preference(context).apply {
    icon = ContextCompat.getDrawable(context,
        R.drawable.ic_menu_camera)
}
```

Preferences API предоставляет набор на основе различных виджетов (помимо текстового). Они часто являются основополагающими для экранов настроек, позволяя пользователям переключать и выбирать различные параметры настроек в приложении. Например можно использовать **CheckBoxPreference**:

```
<CheckBoxPreference
    android:key="checkbox"
    android:title="@string/title_checkbox_preference"
    android:summary="@string/preference_summary"/>
```



Или программно:

```
val checkBoxPreference = CheckBoxPreference(context).apply {
    key = "checkbox"
    title = "Checkbox"
    summary = "This one has a checkbox"
}
```

Также в настройках можно задействовать switch-переключатели:

```
<SwitchPreferenceCompat
    android:key="switch"
    android:title="@string/title_switch_preference"
    android:summary="@string/preference_summary"/>
```

My switch preference
My preference summary



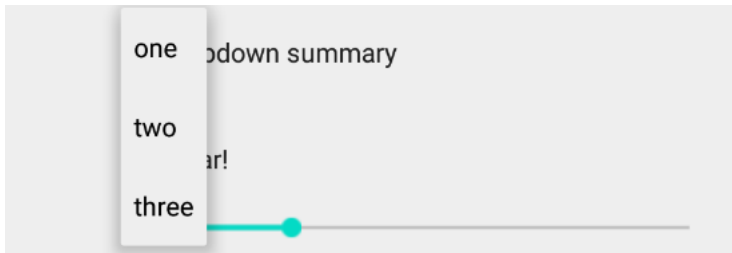
Можно задействовать так называемый DropDownPreference - этот виджет позволяет отображать раскрывающийся список элементов на выбор пользователя. В приложение добавим два ресурс-массива:

```
<string-array name="entries">
    <item>one</item>
    <item>two</item>
    <item>three</item>
</string-array>
<string-array name="entry_values">
    <item>1</item>
    <item>2</item>
    <item>3</item>
</string-array>
```

Тогда код для DropDownPreference может быть следующий:

```
<DropDownPreference
    android:key="dropdown"
    android:title="@string/title_dropdown_preference"
    android:entries="@array/entries"
    app:useSimpleSummaryProvider="true"
    android:entryValues="@array/entry_values"/>
```

При нажатии на элемент настройки появляется выпадающий список.



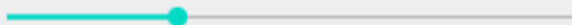
Создание соответствующей настройки программно будет выглядеть следующим образом:

```
val dropDownPreference = DropDownPreference(context).apply {
    key = "drop_down"
    title = "Some title"
    entries = arrayOf("One", "Two", "Three")
    entryValues = arrayOf("1", "2", "3")
}
```

Следующим примером настройки выступает SeekBarPreference, в котором задействована специальная шкала:

```
<SeekBarPreference
    android:key="seekbar"
    android:title="Seek bar!"
    android:max="10"
    android:defaultValue="5" />
```

Seek bar!



В данном случае существует еще несколько атрибутов, которые мы можем установить, кроме обычных ключей и заголовка.

- max - максимальное значение, которое можно выбрать с помощью шкалы поиска.
- defaultValue - значение по умолчанию, которое будет отображаться на панели поиска, когда его нет/до того, как оно будет установлено.

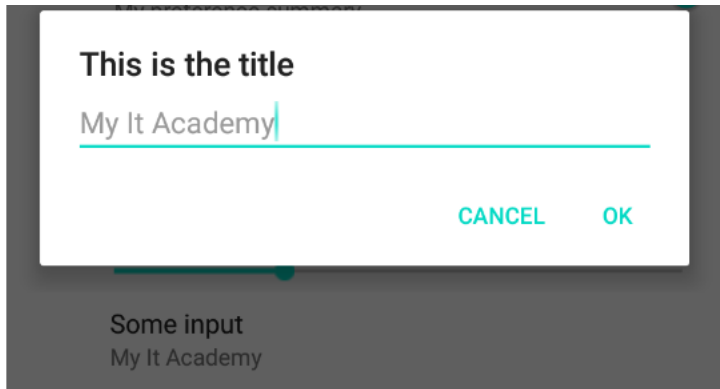
При добавлении данной настройки программно можно использовать следующий код:

```
val seekBarPreference = SeekBarPreference(context).apply {
    key = "seekbar"
    title = "Some title"
    max = 10
    setDefaultValue(5)
}
```

Библиотека AndroidX Preference позволяет использовать элементы настроек, в которых пользователь можно ввести некоторые данные в отдельном окне. Первым из них является элемент EditTextPreference, который позволяет ввести некоторый текст, который будет сохранен в качестве значения для элемента настройки.

```
<EditTextPreference
    android:key="edittext"
    android:title="Some input"
    app:useSimpleSummaryProvider="true"
    android:dialogTitle="This is the title"/>
```

При нажатии на виджет появится следующее диалоговое окно:

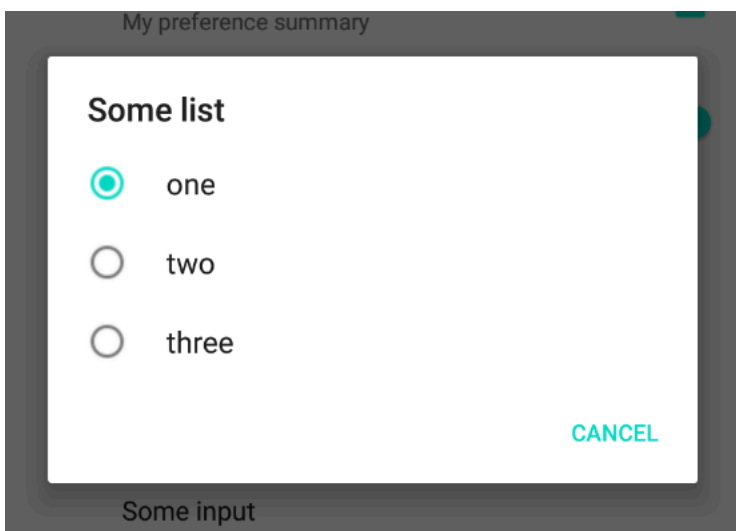


Программно элемент настройки EditTextPreference можно создать следующим образом:

```
val editTextPreference = EditTextPreference(context).apply {
    key = "edit_text"
    title = "Some title"
}
```

Следующим виджетом выступает ListPreference, который отображает предопределенный список элементов:

```
<ListPreference
    android:key="list"
    android:title="Some list"
    app:useSimpleSummaryProvider="true"
    android:entries="@array/entries"
    android:entryValues="@array/entry_values" />
```

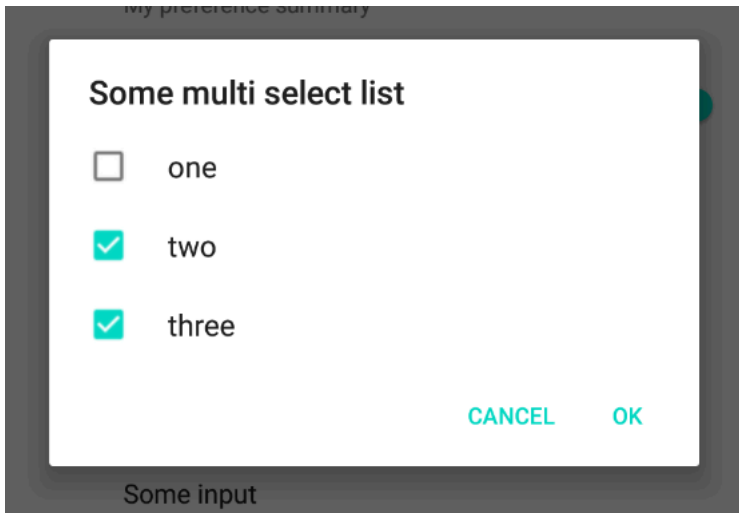


При создании элемента программно, код может выглядеть следующим образом:

```
val listPreference = ListPreference(context).apply {
    key = "drop_down"
    title = "Some title"
    entries = arrayOf("One", "Two", "Three")
    entryValues = arrayOf("1", "2", "3")
}
```

Наконец, элемент `MultiSelectListPreference` работает аналогично элементу `ListPreference` с той разницей, что при этом пользователь может выбрать несколько значений списка.

```
<MultiSelectListPreference
    android:key="multi_select_list"
    android:title="Some multi select list"
    android:summary="This is a summary"
    android:entries="@array/entries"
    android:entryValues="@array/entry_values"/>
```



`MultiSelectListPreference` также можно создать программно следующим образом:

```
val multiSelectListPreference = MultiSelectListPreference(context).apply {
    key = "multi_select_list"
    title = "Some multi select list"
    summary = "This is a summary"
    entries = arrayOf("One", "Two", "Three")
    entryValues = arrayOf("1", "2", "3")
}
```

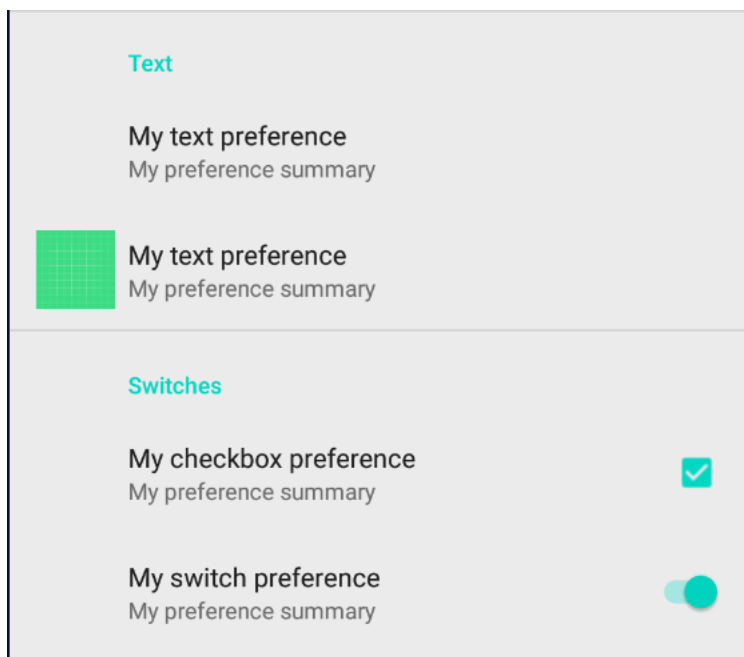
Для группировки настроек используют их разбиение на категории. Например сделать это можно следующим образом:

```
<PreferenceCategory
    android:title="Text">
    <Preference
        android:key="preference_key"
        android:title="@string/preference_title"
        android:summary="@string/preference_summary" />

    <Preference
        android:key="preference_key"
        android:title="@string/preference_title"
        android:summary="@string/preference_summary"
        android:icon="@drawable/ic_launcher_background" />
</PreferenceCategory>
<PreferenceCategory
    android:title="Switches">
    <CheckBoxPreference
        android:key="checkbox"
        android:title="@string/title_checkbox_preference"
        android:summary="@string/preference_summary"/>

    <SwitchPreferenceCompat
        android:key="switch"
        android:title="@string/title_switch_preference"
        android:summary="@string/preference_summary"/>
</PreferenceCategory>
```

На экране приложения это будет выглядеть следующим образом:



PreferenceCategory также может быть создана программно:

```
val notificationCategory = PreferenceCategory(context).apply {  
    key = "notifications_category"  
    title = "Notifications"  
}
```

3. 5.3.3. Отображение настроек

Для отображения экрана настроек приложения, необходимо создать экземпляр класса `PreferenceFragmentCompat`. Если мы собираемся отображать настройки из файла `xml`, то код будет следующим:

```
class SettingsFragment : PreferenceFragmentCompat() {
    override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
        setPreferencesFromResource(R.xml.prefs, rootKey)
    }
}
```

Внутри класса переопределим метод `onCreatePreferences` для настройки нашего экрана `Preferences`. В данном фрагменте можно настроить параметры программно с использованием класса `PreferenceManager`. В случае конфигурации меню настройки программно настраивать элементы необходимо вручную. Например это можно сделать следующим образом:

```
override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
    val context = preferenceManager.context
    val screen = preferenceManager.createPreferenceScreen(context)
    val notificationPreference = SwitchPreferenceCompat(context)
    val feedbackPreference = Preference(context)
    screen.addPreference(notificationPreference)
    screen.addPreference(feedbackPreference)
    preferenceScreen = screen
}
```

`Preference Manager` возвращает экземпляр класса `PreferenceScreen`, который используется для отображения экрана настроек. Для того чтобы виджеты настроек отображались, необходимо задействовать метод `addPreference()`. Его можно вызвать для любого элемента настройки. Сначала необходимо создать элемент настроек, например следующим образом:

```
val notificationCategory = PreferenceCategory(context).apply {
    key = "notifications"
    title = "notifications"
    initialExpandedChildrenCount = 1
}
```

Далее следует передать его методу `addPreference()` и присвоить переменную `screen` переменной `preferenceScreen`:

```
screen.addPreference(notificationPreference)
preferenceScreen = screen
```

Далее можно создать активность для отображения фрагмента с настройками приложения. В приведенном ниже примере создан вложенный класс `SettingsFragment` внутри `SettingsActivity`:

```
class SettingsActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.settings_activity)
        supportFragmentManager
            .beginTransaction()
            .replace(R.id.settings, SettingsFragment())
            .commit()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
            setPreferencesFromResource(R.xml.prefs, rootKey)
        }
    }
}
```

Обратите внимание, что класс `SettingsFragment` расширяет `PreferenceFragment` и имеет метод `setPreferenceFromResource` внутри `onCreate()`. В этом методе передаем идентификатор ресурса `R.xml.prefs` с настройками. Наконец, мы размещаем фрагмент в активности, используя `FragmentManager`.

Другой способ создания активности с настройками - перейти в папку `src` в проекте Android, а нажать правой кнопкой мыши **New > Activity > Settings Activity**.

Для чтения настроек приложения по умолчанию можно задействовать класс `PreferenceManager`. Мы можем получить доступ к значению настроек, используя ключ, который мы назначили ему во время их определения.

```
val sharedPreferences =  
    PreferenceManager.getDefaultSharedPreferences(activity_context)  
val name = sharedPreferences.getString("signature", "")
```

4. 5.3.4. Пример обработки изменения настроек приложения

Рассмотрим [пример](#) обработки изменения настроек приложения.

Создадим простое приложение, состоящее из главной активности, активности с настройками с одним элементом `SwitchPreference`, при изменении которого на экране будет появляться Toast-сообщение со статусом виджета (On/Off).

Класс `MainActivity.kt` будет выглядеть следующим образом:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        val inflater = menuInflater
        inflater.inflate(R.menu.menu, menu)
        return super.onCreateOptionsMenu(menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        // Handle presses on the action bar menu items
        when (item.itemId) {
            R.id.action_settings -> {
                val intent = Intent(this, SettingsActivity::class.java)
                startActivity(intent)
                return true
            }
        }
        return super.onOptionsItemSelected(item)
    }
}
```

Далее создадим разметку для меню главной активности. Нажмем правой кнопкой мыши в проекте на каталог `res` и далее в открывшемся списке выберем пункт **New -> Android Resource File**. Далее в появившемся окне укажем для имени файла название (например, `menu`), а для поля **Resource Type** (тип ресурса) выберем **Menu**. Добавим туда следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="1"
        android:title="@string/settings" />
</menu>
```

Далее создадим `SettingsActivity.kt`. При создании использует автоматический способ создания. Перейдем в папку `src` в проекте Android, а нажать правой кнопкой мыши **New -> Activity -> Settings Activity**. В итоге создаться активность со следующим кодом:

```
import android.os.Bundle
import android.preference.PreferenceFragment
import android.support.v7.app.AppCompatActivity

class SettingsActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.settings_activity)
        if (savedInstanceState == null) {
            supportFragmentManager
                .beginTransaction()
                .replace(R.id.settings, SettingsFragment())
                .commit()
        }
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
            setPreferencesFromResource(R.xml.root_preferences, rootKey)
        }
    }
}
```

Изменим описание `SettingsActivity` файле манифеста проекта следующим образом:

```
<activity android:name=".SettingsActivity"
    android:label="Settings"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```

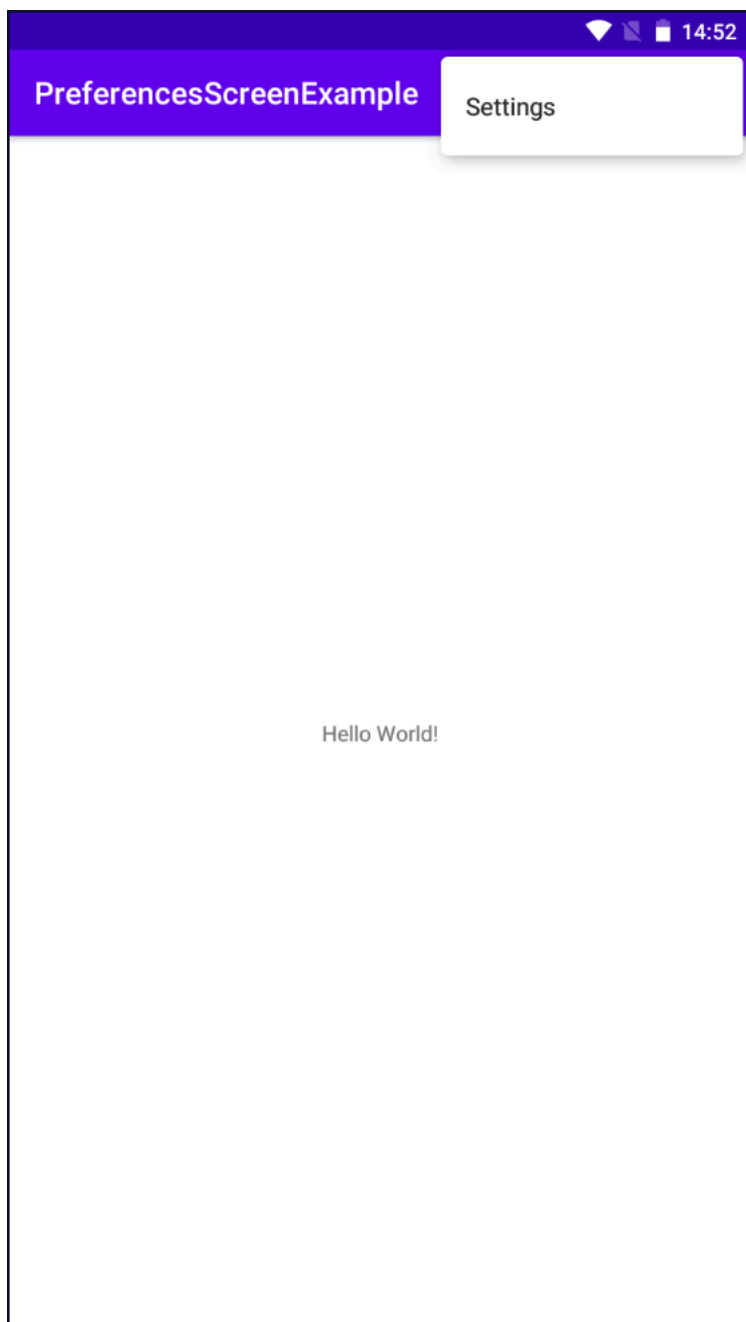
Далее изменим файл `root_preferences.xml` так, чтобы в нем содержался один элемент `SwitchPreference`:

```
<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <SwitchPreference
        android:defaultValue="true"
        android:key="example_switch"
        android:summary="Turn this option on or off"
        android:title="Settings option" />
</PreferenceScreen>
```

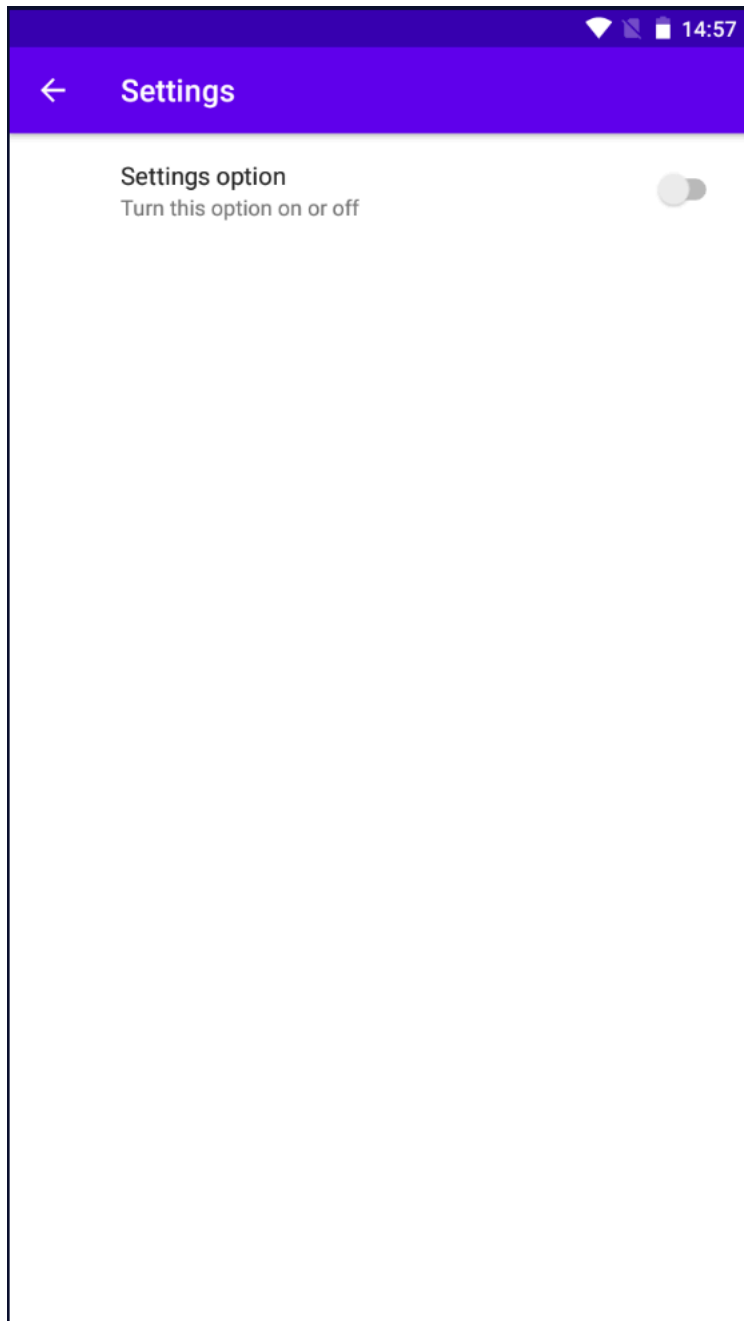
Для работы с AndroidX Preferences не забудьте добавить необходимые зависимости в файл `build.gradle`:

```
implementation 'androidx.preference:preference:1.1.1'
```

Запустим приложение. При нажатии на меню в главной активности приложения будет показан список из одного элемента "Settings".



При нажатии пункта меню открывается SettingsActivity:



Одним из способов отследить изменение настроек приложения выступает использование интерфейса [OnSharedPreferenceChangeListener](#). Он состоит из метода `onSharedPreferenceChanged`, который вызывается при изменении, добавлении или удалении общих настроек и выглядит следующим образом:

```
abstract fun onSharedPreferenceChanged(  
    sharedPreferences: SharedPreferences!,  
    key: String!  
): Unit
```

Реализуем названный интерфейс в классе SettingsActivity:

```
class SettingsActivity : AppCompatActivity(), SharedPreferences.OnSharedPreferenceChangeListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.settings_activity)
        if (savedInstanceState == null) {
            supportFragmentManager
                .beginTransaction()
                .replace(R.id.settings, SettingsFragment())
                .commit()
        }
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
    }

    class SettingsFragment : PreferenceFragmentCompat() {
        override fun onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {
            setPreferencesFromResource(R.xml.root_preferences, rootKey)
        }
    }

    override fun onSharedPreferenceChanged(sharedPreferences: SharedPreferences?, key: String?) {
    }
}
```

Для того, чтобы следить за изменениями настроек в приложении сначала в 'onCreate()' создадим объект SharedPreferences с помощью метода `getDefaultSharedPreferences()` из объекта `PreferenceManager` и зарегистрируем его для отслеживания изменений настроек:

```
val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this)
sharedPreferences.registerOnSharedPreferenceChangeListener(this)
```

Переопределим метод `onDestroy()` в `SettingsActivity` и снимем регистрацию для отслеживания изменений настроек:

```
override fun onDestroy() {
    val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this)
    sharedPreferences.unregisterOnSharedPreferenceChangeListener(this)
    super.onDestroy()
}
```

Регистрацию и снятие регистрации изменения настроек можно также добавить, переопределив методы `onResume()` и `onPause()` соответственно.

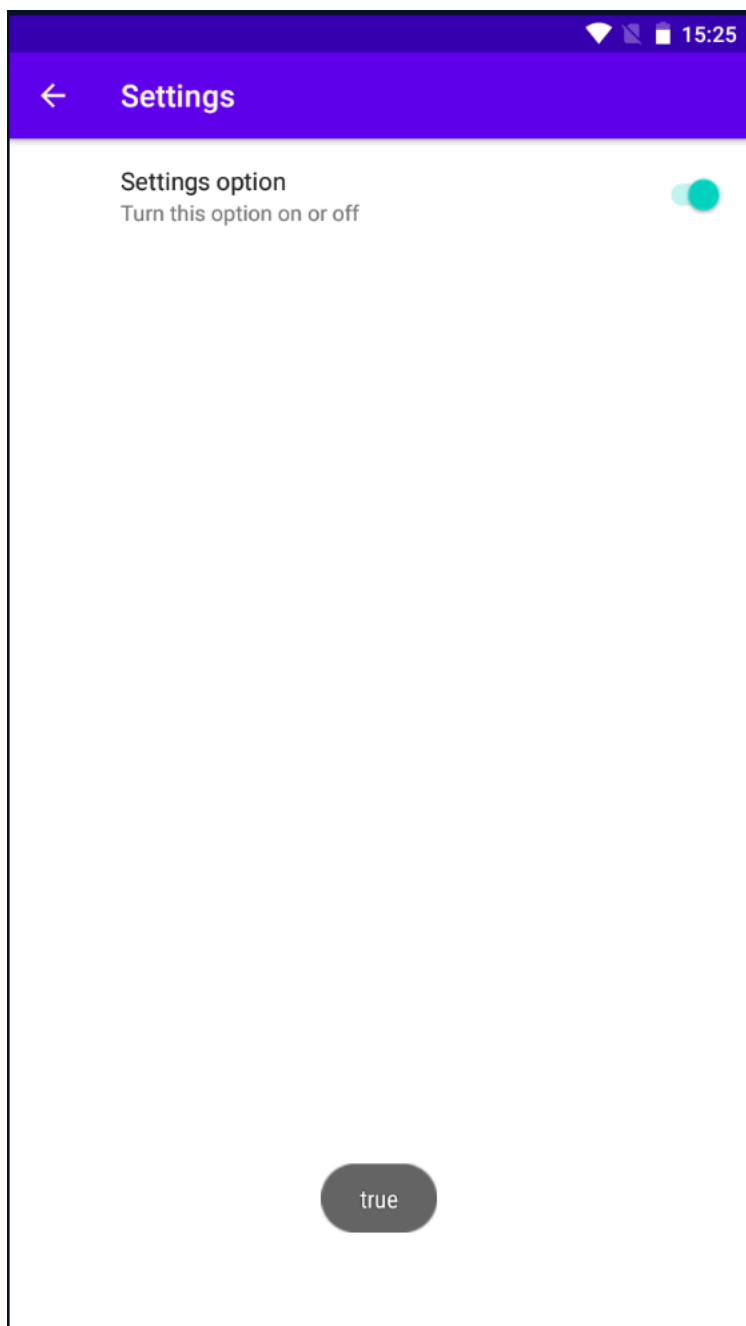
```
override fun onResume() {
    super.onResume()
    PreferenceManager.getDefaultSharedPreferences(this).registerOnSharedPreferenceChangeListener(this)
}

override fun onPause() {
    super.onPause()
    PreferenceManager.getDefaultSharedPreferences(this).unregisterOnSharedPreferenceChangeListener(this)
}
```

Теперь реализуем задуманный функционал приложения. В метод `onSharedPreferenceChanged` класса `SettingsActivity` добавим следующий код:

```
when (key) {
    "example_switch" -> {
        val test = sharedPreferences!!.getBoolean("example_switch", false)
        Toast.makeText(this, test.toString(), Toast.LENGTH_SHORT).show()
    }
}
```

В данном методе по ключу `example_switch` мы получаем значение элемента `SwitchPreference` и отображаем его в Toast-сообщении.



Теперь вы можете самостоятельно добавит ь остальные виды настроек и отследить их изменения.

[Начать тур для пользователя на этой странице](#)