**Тема 13**. Налаштування. Оброблення даних з соціальних мереж та Cloud Storage. Платформа доступу до сховищ

AndroidManifest.xml – це потужний файл на платформі Android, який дозволяє описати функціональні можливості та вимоги додатків до Android. Тим не менш, робота з ним не є простою. Xamarin.Android допомагає звести до мінімуму цю труднощі, дозволяючи додавати атрибути, що налаштовуються, в класи, які потім будуть використовуватися для автоматичного створення маніфесту. Наша мета полягає в тому, що 99% користувачів ніколи не повинні змінювати AndroidManifest.xml вручну.

AndroidManifest.xml створюється як частина процесу складання, а XML-код, знайдений у властивостях або AndroidManifest.xml , об'єднується з XML, створеним з атрибутів користувача. Результуючий об'єднаний AndroidManifest.xml знаходиться в підкаталозі obj; наприклад, він знаходиться в obj/Debug/android/AndroidManifest.xml для збирання налагодження. Процес злиття простий: він використовує атрибути, що настроюються в коді для створення XML-елементів і вставляє ці елементи в AndroidManifest.xml.

Під час компіляції збірки перевіряються на наявність класів, що abstract не є похідними від дії, і [Activity] атрибут оголошується на них. Потім для створення маніфесту використовуються ці класи та атрибути. Розглянемо наступний приклад коду:

```
namespace Demo
{
   public class MyActivity : Activity
   {
   }
}
```

Це призводить до того, що в AndroidManifest.xml не створюється нічого. Якщо потрібно <activity/> створити елемент, необхідно використовувати [Activity] атрибут, що настроюється:

```
namespace Demo
```

```
{
    [Activity]
    public class MyActivity : Activity
    {
    }
}
```

Цей приклад призводить до додавання наступного фрагмента XML до AndroidManifest.xml:

```
<activity android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity" />
```

Атрибут [Activity] не впливає на типи abstract; abstract типи ігноруються.

**Activity Name**

Починаючи з версії Xamarin.Android 5.1, ім'я типу дії засноване на імені MD5SUM імені експортованого типу. Це дозволяє надати те саме повне ім'я з двох різних збірок і не отримати помилку упаковки. (До Xamarin.Android 5.1 ім'я типу дії за умовчанням було створено з нижнього регістру простору імен та імені класу.)

Якщо ви бажаєте перевизначити це значення за умовчанням і явно вказати ім'я дії, використовуйте Name властивість:

```
[Activity (Name="awesome.demo.activity")]
public class MyActivity : Activity
{
}
```

У цьому прикладі створюється наступний фрагмент XML:

```
<activity android:name="awesome.demo.activity" />
```

Властивість слід використовувати Name лише з причин зворотної сумісності, оскільки таке перейменування може уповільнити пошук типів під час виконання. Якщо у вас є застарілий код, який очікує, що ім'я типу за промовчанням дії буде засноване на нижньому регістрі простору імен та імені класу, див. статтю Про іменування викликаного оболонки Android, щоб отримати поради щодо підтримки сумісності.

**Рядок заголовка дії**

За промовчанням Android надає програмі рядок заголовка під час запуску. Для цього використовується /manifest/application/activity/@android:labelзначення . У більшості випадків це значення відрізнятиметься від імені класу. Щоб вказати мітку програми у заголовку рядка, використовуйте Label властивість. Наприклад:

```
[Activity (Label="Awesome Demo App")]
public class MyActivity : Activity
{
}
```

У цьому прикладі створюється наступний фрагмент XML:

```
<activity android:label="Awesome Demo App"
android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity" />
```

**Доступний для запуску з програми вибору програм**

За промовчанням ваша дія не відображатиметься на екрані засобу запуску програм Android. Це пов'язано з тим, що додаток, швидше за все, буде багато дій, і ви не хочете значок для кожного з них. Щоб вказати, який з них повинен бути запущений із запуску програм, використовуйте MainLauncher властивість. Наприклад:

```
[Activity (Label="Awesome Demo App", MainLauncher=true)]
public class MyActivity : Activity
{
}
```

У цьому прикладі створюється наступний фрагмент XML:

```
<activity android:label="Awesome Demo App"
android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

**Значок дії**

За промовчанням ваша дія буде присвоєна значку запуску за промовчанням, наданим системою. Щоб використовувати значок користувача, спочатку додайте .png в ресурси або малювання, задайте для властивості дію збірки AndroidResource, а потім використовуйте Icon властивість, щоб вказати значок, що використовується. Наприклад:

```
[Activity (Label="Awesome Demo App", MainLauncher=true, Icon="@drawable/myicon")]
public class MyActivity : Activity
{
}
```

У цьому прикладі створюється наступний фрагмент XML:

```
<activity android:icon="@drawable/myicon" android:label="Awesome Demo App"
        android:name="md5a7a3c803e481ad8926683588c7e9031b.MainActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

**Дозволи**

При додаванні дозволів на маніфест Android (як описано в розділі "Додавання дозволів на маніфест Android"), ці дозволи записуються у властивості або AndroidManifest.xml. Наприклад, якщо ви задали INTERNET дозвіл, наступний елемент додається в properties/AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Налагоджувальні збірки автоматично задають деякі дозволи для спрощення налагодження (наприклад INTERNET , та READ_EXTERNAL_STORAGE) — ці параметри задаються лише у створеному obj/debug/android/AndroidManifest.xml і не відображаються як увімкнені у параметрах необхідних дозволів .

Наприклад, при перевірці створеного файлу маніфесту в obj/Debug/android/AndroidManifest.xml можуть відображатися такі додані елементи дозволів:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

У версії збірки випуску маніфесту (obj/Debug/android/AndroidManifest.xml) ці дозволи не налаштовуються автоматично. Якщо ви виявите, що перемикання на збірку випуску призводить до втрати програми дозволу, доступної у збиранні налагодження, переконайтеся, що ви явно задали цю дозвіл у параметрах необхідних дозволів для програми (див. розділ "Складання > програми Android у Visual Studio для Mac; див. розділ "Властивості > маніфесту Android" у Visual Studio).

**Cloud Storage**

Cloud Storage for Firebase – це потужна, проста та економічна служба зберігання об'єктів, створена для масштабу Google. Firebase SDK для Cloud Storage додає безпеку Google для завантаження файлів для ваших програм Firebase, незалежно від якості мережі. SDK Firebase можна використовувати для зберігання зображень, аудіо, відео або іншого контенту користувача.

Cloud Storage for Firebase створено для розробників додатків, яким необхідно зберігати та обслуговувати контент користувача, наприклад фотографії або відео.

Основні можливості:

Надійні операції. Пакети Cloud Storage виконують завантаження та вивантаження незалежно від якості мережі. Завантаження та вивантаження надійні, тобто вони перезапускаються з того місця, де зупинилися, що заощаджує час та пропускну спроможність користувачів.

Сильна безпека. Пакети Cloud Storage інтегруються з Firebase Authentication, щоб забезпечити просту та інтуїтивно зрозумілу автентифікацію

для розробників. Можна використовувати декларативну модель безпеки, щоб дозволити доступ на основі імені файлу, розміру, типу вмісту та інших метаданих.

Висока масштабованість. Cloud Storage створено для ексабайтного масштабу, коли програма стає «вірусною». Легко перейти від прототипу до виробництва, використовуючи ту ж саму інфраструктуру, що і Spotify та Google Фото.

Cloud Storage зберігає файли в кошику Google Cloud Storage, що робить їх доступними як через Firebase, так і через Google Cloud. Це дозволяє гнучко завантажувати та вивантажувати файли з мобільних клієнтів через Cloud Storage for Firebase SDK. Крім того, можна виконувати обробку на стороні сервера, наприклад фільтрацію зображень або транскодування відео, використовуючи API-інтерфейси Google Cloud Storage. Cloud Storage масштабується автоматично, а це означає, що немає необхідності переходити до іншого постачальника.