Бесплатная электронная книга

# УЧУСЬ

# Xamarin.Android

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#xamarin.an

droid

# Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: xamarin-android

It is an unofficial and free Xamarin.Android ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Xamarin.Android.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# глава 1: Начало работы с Xamarin.Android

## замечания

Xamarin.Android позволяет создавать собственные приложения для Android, используя те же элементы управления пользовательского интерфейса, что и в Java, за исключением гибкости и элегантности современного языка (C #), возможностей библиотеки базового класса .NET (BCL) и двух первоклассные IDE - Xamarin Studio и Visual Studio - у них под рукой.

Для получения дополнительной информации об установке Xamarin.Android на компьютере Mac или Windows обратитесь к руководствам « Начало работы» в центре разработчика Xamarin

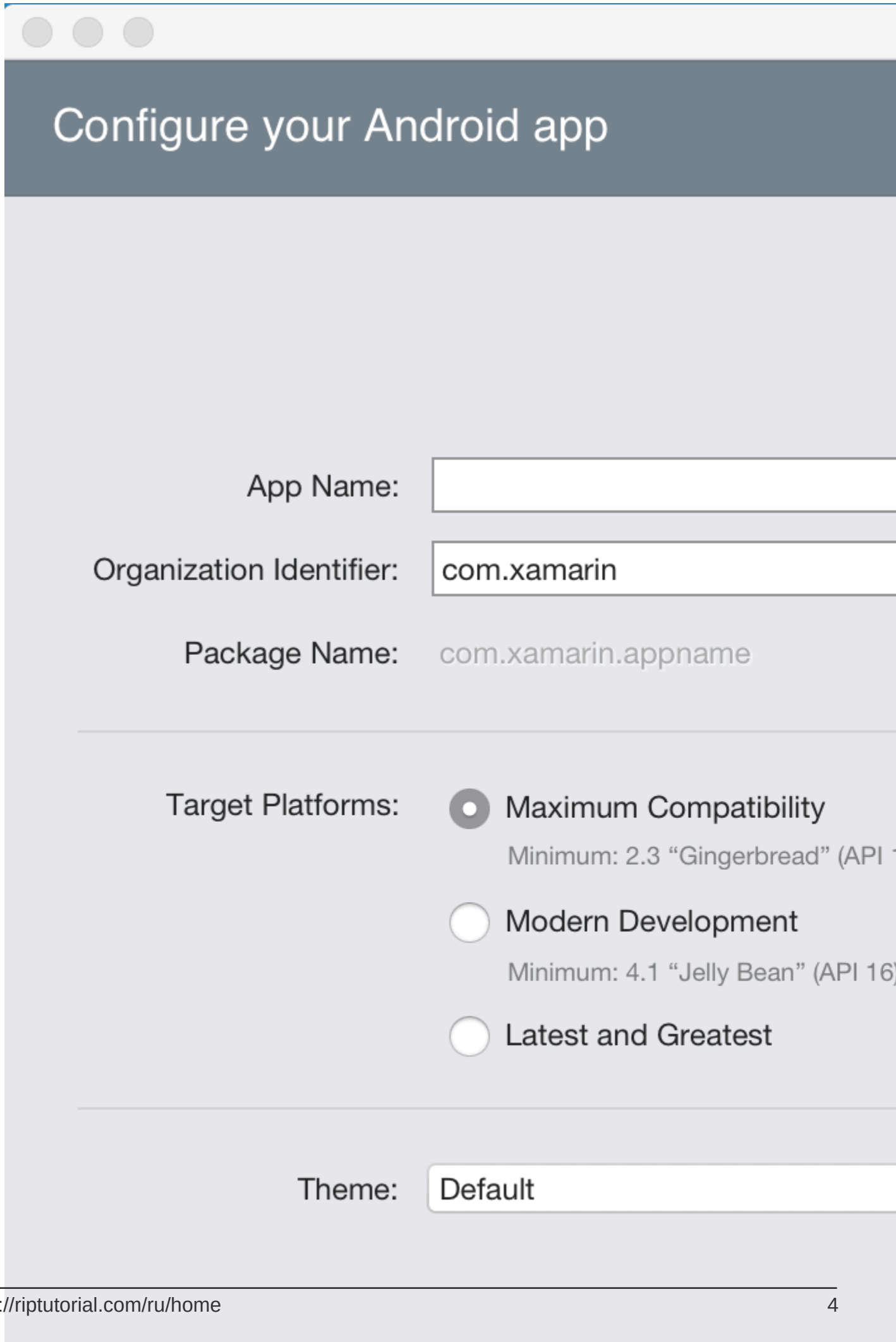
## Версии

| Версия | Кодовое имя | Уровень API | Дата выхода |
|---|---|---|---|
| 1,0 | Никто | 1 | 2008-09-23 |
| 1,1 | Никто | 2 | 2009-02-09 |
| 1,5 | Кекс | 3 | 2009-04-27 |
| 1,6 | Пончик | 4 | 2009-09-15 |
| 2,0-2,1 | Эклер | 5-7 | 2009-10-26 |
| 2.2-2.2.3 | Froyo | 8 | 2010-05-20 |
| 2.3-2.3.7 | Имбирный пряник | 9-10 | 2010-12-06 |
| 3.0-3.2.6 | ячеистый | 11-13 | 2011-02-22 |
| 4.0-4.0.4 | Сэндвич с мороженым | 14-15 | 2011-10-18 |
| 4.1-4.3.1 | Жевательные конфеты | 16-18 | 2012-07-09 |
| 4.4-4.4.4, 4.4W-4.4W.2 | Кит-Кат | 19-20 | 2013-10-31 |
| 5.0-5.1.1 | леденец | 21-22 | 2014-11-12 |
| 6.0-6.0.1 | зефирка | 23 | 2015-10-05 |
| 7,0 | нуга | 24 | 2016-08-22 |

# Examples

## Начало работы в Xamarin Studio

1. Найдите **Файл> Создать> Решение,** чтобы открыть новый диалог проекта.
2. Выберите **приложение Android** и нажмите « **Далее»** .
3. Настройте приложение, установив имя приложения и идентификатор организации. Выберите целевую платформу, наиболее подходящую для ваших нужд, или оставьте ее как значение по умолчанию. Нажмите "Далее:

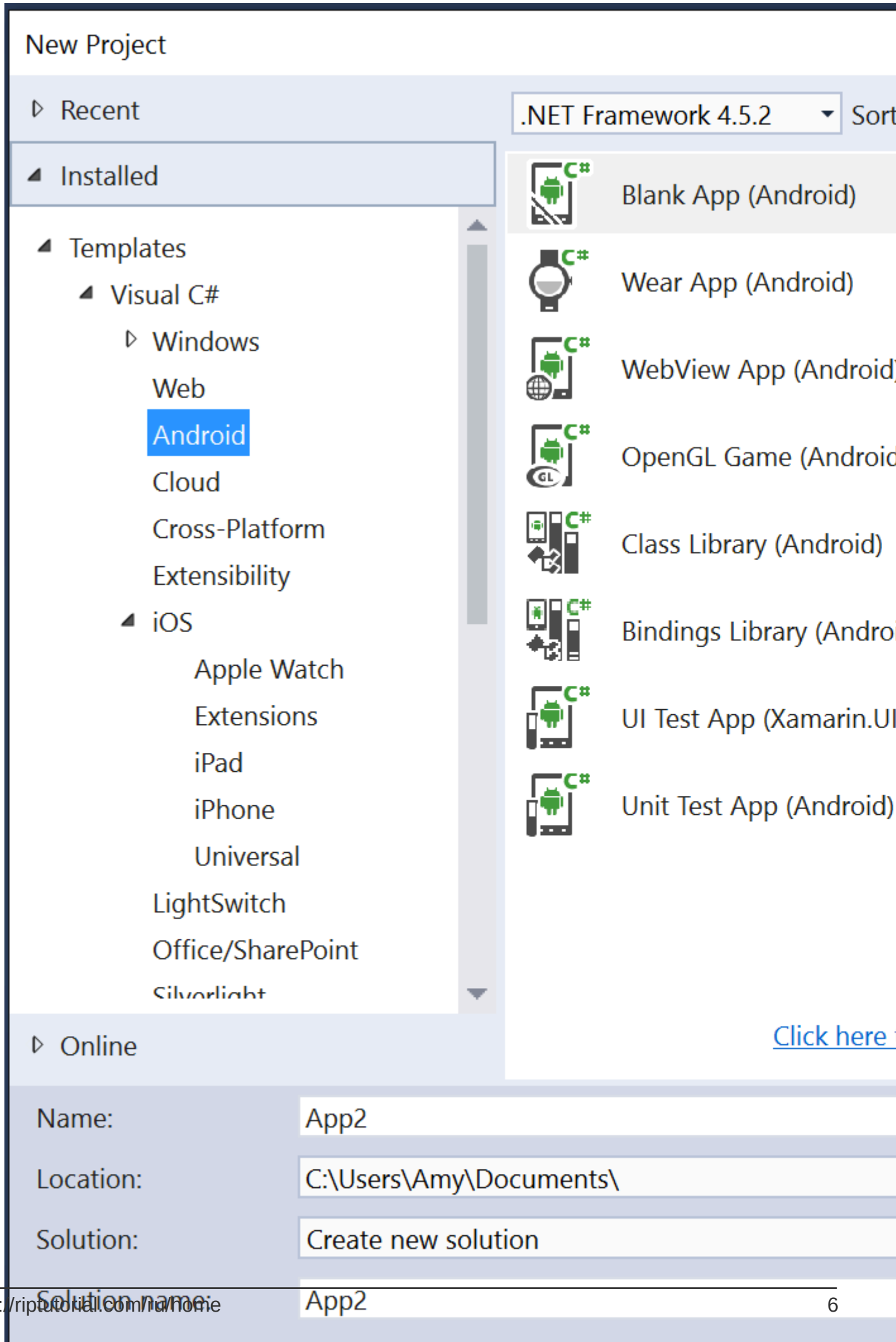

Configure your Android app
App Name:
Organization Identifier:
com.xamarin
Package Name:
com.xamarin.appname
Target Platforms:
Maximum Compatibility
Minimum: 2.3 “Gingerbread” (API
Modern Development
Minimum: 4.1 “Jelly Bean” (API 16)
Latest and Greatest
Theme:
Default

4. Задайте название проекта и имя решения или оставьте в качестве имени по умолчанию. Нажмите «Создать», чтобы создать проект.
5. Настройте устройство для развертывания или настройте эмулятор
6. Чтобы запустить приложение, выберите «Конфигурация **отладки»** и нажмите кнопку «Воспроизведение»:

и нажмите кнопку «Воспроизведение»:


## Начало работы в Visual Studio

1. Перейдите в **меню «Файл»> «Создать»> «Проект»,** чтобы открыть диалог «Новый проект».
2. Перейдите к **Visual C #> Android** и выберите Blank App:


New Project
Recent
.NET Framework 4.5.2
Sort
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
Cross-Platform
Extensibility
iOS
Apple Watch
Extensions
iPad
iPhone
Universal
LightSwitch
Office/SharePoint
Silverlight
Blank App (Android)
Wear App (Android)
WebView App (Android)
OpenGL Game (Android
Class Library (Android)
Bindings Library (Andro
UI Test App (Xamarin.UI
Unit Test App (Android)
Online
Click here
Name:
App2
Location:
C:\Users\Amy\Documents\
Solution:
Create new solution
Solution name:
App2

3. Дайте вашему приложению **имя** и нажмите **OK,** чтобы создать проект.
4. Настройте устройство для развертывания или настройте эмулятор
5. Чтобы запустить приложение, выберите конфигурацию **отладки** и нажмите кнопку «

**Пуск»** :

**Пуск»** : 


Прочитайте Начало работы с Xamarin.Android онлайн: https://riptutorial.com/ru/xamarin-android/topic/403/начало-работы-с-xamarin-android

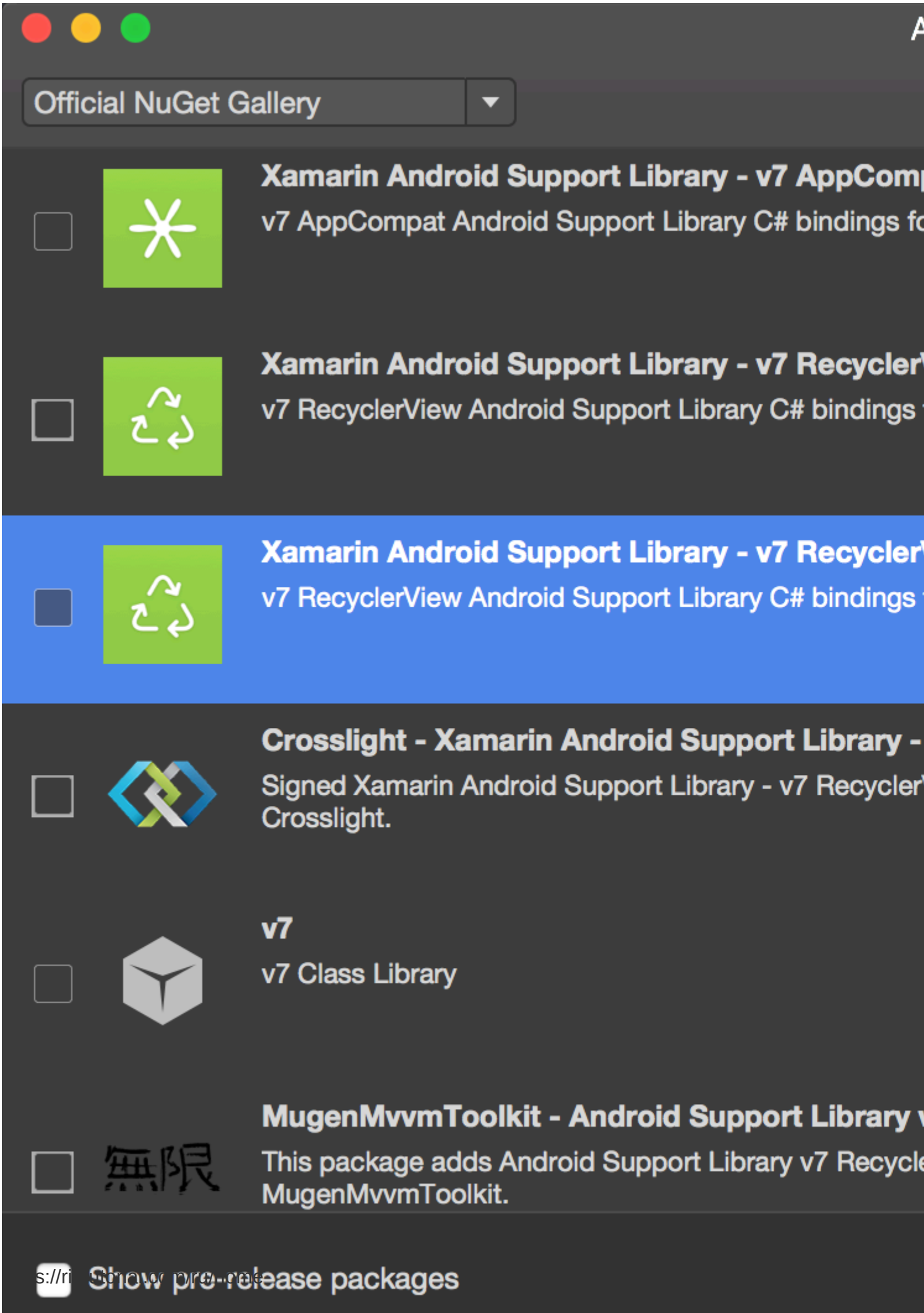
# глава 2: RecyclerView

## Examples

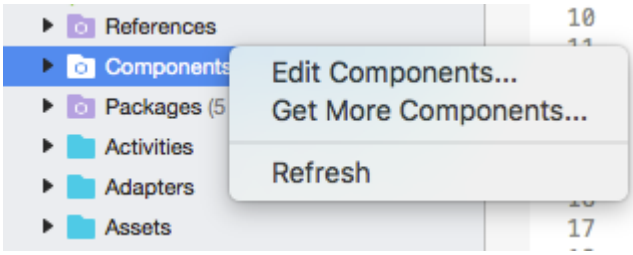
### Основы RecyclerView

Это пример использования `Android Support Library V7 RecyclerView` . Библиотеки поддержки обычно рекомендуются, поскольку они обеспечивают обратные совместимые версии новых функций, предоставляют полезные элементы пользовательского интерфейса, которые не входят в структуру, и предоставляют ряд утилит, которые приложения могут использовать.

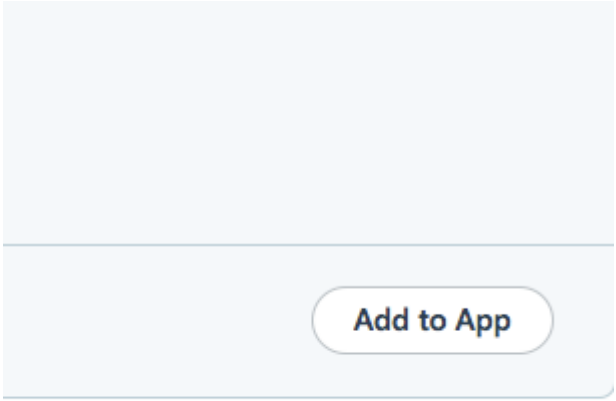
Чтобы получить `RecyclerView` , мы установим необходимые пакеты Nuget. Сначала мы будем искать `v7 recyclerview` . Прокрутите вниз, пока мы не увидим `Xamarin Android Support Library - v7 RecyclerView` . Выберите его и нажмите « **Добавить пакет»** .


Official NuGet Gallery
Xamarin Android Support Library - v7 AppCom
v7 AppCompat Android Support Library C# bindings fo
Xamarin Android Support Library - v7 Recycler
v7 RecyclerView Android Support Library C# bindings
Xamarin Android Support Library - v7 Recycler
v7 RecyclerView Android Support Library C# bindings
Crosslight - Xamarin Android Support Library -
Signed Xamarin Android Support Library - v7 Recycler
Crosslight.
v7
v7 Class Library
MugenMvvmToolkit - Android Support Library
This package adds Android Support Library v7 Recycle
MugenMvvmToolkit.
Show pre-release packages

Xamarin. Чтобы добавить компонент, щелкните правой кнопкой мыши « `Components` в проекте Android в обозревателе решений и нажмите « `Get More Components` .




В появившемся окне «Магазин компонентов» найдите RecyclerView. В списке поиска выберите `Android Support Library V7 RecyclerView` . Затем нажмите « `Add to App` . Компонент добавляется в проект.




Следующий шаг - добавить RecyclerView на страницу. В `axml` (layout) мы можем добавить `RecyclerView` как `RecyclerView` ниже.

```
<android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:scrollbars="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

RecyclerView требует, чтобы как минимум два вспомогательных класса были настроены для базовой стандартной реализации: `Adapter` и `ViewHolder` . `Adapter` раздувает макеты элементов и связывает данные с представлениями, отображаемыми в RecyclerView. ViewHolder просматривает и сохраняет ссылки на просмотр. Владелец просмотра также помогает при обнаружении кликов по элементам.

Вот базовый пример класса адаптера

```
public class MyAdapter : RecyclerView.Adapter
{
    string [] items;

    public MyAdapter (string [] data)
    {
```

```
        items = data;
    }

    // Create new views (invoked by the layout manager)
    public override RecyclerView.ViewHolder OnCreateViewHolder (ViewGroup parent, int
viewType)
    {
        // set the view's size, margins, paddings and layout parameters
        var tv = new TextView (parent.Context);
        tv.SetWidth (200);
        tv.Text = "";

        var vh = new MyViewHolder (tv);
        return vh;
    }

    // Replace the contents of a view (invoked by the layout manager)
    public override void OnBindViewHolder (RecyclerView.ViewHolder viewHolder, int position)
    {
        var item = items [position];

        // Replace the contents of the view with that element
        var holder = viewHolder as MyViewHolder;
        holder.TextView.Text = items[position];
    }

    public override int ItemCount {
        get {
            return items.Length;
        }
    }
}
```

В методе `OnCreateViewHolder` мы сначала надуваем представление и создаем экземпляр класса ViewHolder. Этот экземпляр должен быть возвращен. Этот метод вызывается адаптером, когда требуется новый экземпляр ViewHolder. Этот метод не будет вызываться для каждой отдельной ячейки. Когда RecyclerView имеет достаточно ячеек для заполнения представления, он будет повторно использовать старые ячейки, которые прокручиваются из представления для дальнейших ячеек.

Для отображения данных в указанной позиции обратный вызов `OnBindViewHolder` вызывается адаптером. Этот метод должен обновлять содержимое itemView, чтобы отразить элемент в данной позиции.

Поскольку ячейка содержит только один `TextView` , мы можем иметь простой ViewHolder, как показано ниже.

```
public class MyViewHolder : RecyclerView.ViewHolder
{
    public TextView TextView { get; set; }


    public MyViewHolder (TextView v) : base (v)
    {
        TextView = v;
```

```
    }
}
```

Следующий шаг - связать вещи с `Activity` .

```
RecyclerView mRecyclerView;
MyAdapter mAdapter;
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);
    SetContentView (Resource.Layout.Main);
    mRecyclerView = FindViewById<RecyclerView> (Resource.Id.recyclerView);

    // Plug in the linear layout manager:
    var layoutManager = new LinearLayoutManager (this) { Orientation =
LinearLayoutManager.Vertical };
    mRecyclerView.SetLayoutManager (layoutManager);
    mRecyclerView.HasFixedSize = true;

    var recyclerViewData = GetData();
    // Plug in my adapter:
    mAdapter = new MyAdapter (recyclerViewData);
    mRecyclerView.SetAdapter (mAdapter);
}

string[] GetData()
{
     string[] data;
     .
     .
     .
     return data;
}
```

Класс LayoutManager отвечает за измерение и позиционирование позиций элементов в RecyclerView, а также за определение политики для того, когда следует перерабатывать элементы, которые больше не видны пользователю. Перед `RecyclerView` нам пришлось использовать `ListView` для размещения ячеек, как в списке вертикальной прокрутки, так и в `GridView` для отображения элементов в двумерной прокручиваемой сетке. Но теперь мы можем добиться как с RecyclerView, установив другой LayoutManger. `LinearLayoutManager` упорядочивает ячейки, как в ListView, а `GridLayoutManager` упорядочивает ячейки сеткой.

## RecyclerView с событиями Click

В этом примере показано, как установить Click EventHandlers в Xamarin.Android RecyclerView.

**В Android Java** для настройки прослушивателя нажмите кнопку onClickListener для просмотра, которое будет нажато, например:

```
ImageView picture = findViewById(R.id.item_picture);
picture.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
```

```
        // do stuff
    }
});
```

**Однако в Xamarin.Android** способ настроить слушателя для события Click - это **добавить** EventHandler следующими способами:

**1.**

```
ImageView picture = FindViewById<ImageView>(Resource.Id.item_picture);
picture.Click += delegate {
    // do stuff
};
```

**2.**

```
ImageView picture = FindViewById<ImageView>(Resource.Id.item_picture);
picture.Click += async delegate {
    // await DoAsyncMethod();
    // do async stuff
};
```

**3.**

```
ImageView picture = FindViewById<ImageView>(Resource.Id.item_picture);
picture.Click += Picture_Click;
... // rest of your method

private void Picture_Click(object sender, EventArgs e)
{
    // do stuff
}
```

Обратите внимание, что **EventHandler добавляется, а не устанавливается.** Если ClickTimeHandler добавлен внутри метода GetView из адаптера GridView / ListView или метода OnBindViewHolder из RecyclerView.Adapter, каждый раз при создании представления элемента будет добавлен новый EventHandler. После прокрутки несколько раз будет добавлено несколько EventHandlers, и при щелчке на экране все они будут запущены.

Чтобы избежать этой проблемы, EventHandlers должны быть отписаны и подписаны впоследствии в методах GetView или OnBindViewHolder. Кроме того, они должны использовать способ номер **3.** для установки EventHandler, иначе невозможно отменить подписку на EventHandlers.

Пример события RecyclerView.Adapter with Click показан ниже:

```
public class ViewHolderPerson : Android.Support.V7.Widget.RecyclerView.ViewHolder
{
    public View Item { get; private set; }
```

```
    public ImageView Picture { get; private set; }
    public TextView Name { get; private set; }

    public ViewHolderPerson(View itemView) : base(itemView)
    {
        this.Item = itemView;
        this.Picture = itemView.FindViewById<ImageView>(Resource.Id.Item_Person_Picture);
        this.Name = itemView.FindViewById<TextView>(Resource.Id.Item_Person_Name);
    }
}

public class AdapterPersons : Android.Support.V7.Widget.RecyclerView.Adapter
{
    private Context context;
    private Android.Support.V7.Widget.RecyclerView recyclerView;
    private List<Person> persons;

    public AdapterPersons(Context context, Android.Support.V7.Widget.RecyclerView
recyclerView, List<Person> persons)
    {
        this.context = context;
        this.recyclerView = recyclerView;
        this.persons = persons;
    }

    public override int ItemCount => persons.Count;

    public override void OnBindViewHolder(RecyclerView.ViewHolder holder, int position)
    {
        Person person = this.persons[position];
        ((ViewHolderPerson)holder).Name.Text = person.Name;
        ((ViewHolderPerson)holder).Picture.SetImageBitmap(person.Picture);

        // Unsubscribe and subscribe the method, to avoid setting multiple times.
        ((ViewHolderPerson)holder).Item.Click -= Person_Click;
        ((ViewHolderPerson)holder).Item.Click += Person_Click;
    }

    private void Person_Click(object sender, EventArgs e)
    {
        int position = this.recyclerView.GetChildAdapterPosition((View)sender);
        Person personClicked = this.persons[position];
        if(personClicked.Gender == Gender.Female)
        {
            Toast.MakeText(this.context, "The person clicked is a female!",
ToastLength.Long).Show();
        }
        else if(personClicked.Gender == Gender.Male)
        {
            Toast.MakeText(this.context, "The person clicked is a male!",
ToastLength.Long).Show();
        }
    }

    public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType)
    {
        View itemView =
LayoutInflater.From(parent.Context).Inflate(Resource.Layout.item_person, parent, false);
        return new ViewHolderPerson(itemView);
    }
```

```
}
```

Прочитайте RecyclerView онлайн: https://riptutorial.com/ru/xamarin-android/topic/3452/recyclerview

# глава 3: Xamarin.Android - Bluetooth-связь

## Вступление

В **Xamarin.Android** свойства **BluetoothSocket.InputStream** и **BluetoothSocket.OutputStream** автоматически преобразуются в **System.IO.Stream** . В случае так называемого интерактивного протокола связи, когда сервер отвечает только тогда, когда клиент общается с ним, System.IO.Stream не подходит, потому что у него нет метода или свойства для получения количества доступных байтов ответа перед чтением ответа.

## параметры

| параметр | подробности |
|---|---|
| разъем | Экземпляр объекта BluetoothSocket. Перед вызовом этого метода необходимо открыть сокет. |
| CMD | Команда в качестве байтового массива для отправки на устройство BT. |
| _mx | Поскольку этот метод использует аппаратный ресурс, лучше назвать его из отдельного рабочего потока. Этот параметр является экземпляром объекта System.Threading.Mutex и используется для синхронизации потока с другими потоками, необязательно вызывающего этот метод. |
| Тайм-аут | Время ожидания в миллисекундах между операциями записи и чтения. |

## Examples

### Отправлять и получать данные с устройства bluetooth с помощью сокета

В приведенном ниже примере используются типы Android.Runtime.InputStreamInvoker и Android.Runtime.OutputStreamInvoker получают Java.IO.InputStream и Java.IO.OutputStream . Когда у нас есть экземпляр **Java.IO.InputStream** , мы можем использовать его **метод . Available ()** для получения количества доступных байтов ответа, которые мы можем использовать в **методе .Read ()** :

```
byte[] Talk2BTsocket(BluetoothSocket socket, byte[] cmd, Mutex _mx, int timeOut = 150)
{
    var buf = new byte[0x20];

    _mx.WaitOne();
    try
```

```
    {
        using (var ost = socket.OutputStream)
        {
            var _ost = (ost as OutputStreamInvoker).BaseOutputStream;
            _ost.Write(cmd, 0, cmd.Length);
        }

        // needed because when skipped, it can cause no or invalid data on input stream
        Thread.Sleep(timeOut);

        using (var ist = socket.InputStream)
        {
            var _ist = (ist as InputStreamInvoker).BaseInputStream;
            var aa = 0;
            if ((aa = _ist.Available()) > 0)
            {
                var nn = _ist.Read(buf, 0, aa);
                System.Array.Resize(ref buf, nn);
            }
        }
    }
    catch (System.Exception ex)
    {
        DisplayAlert(ex.Message);
    }
    finally
    {
        _mx.ReleaseMutex();     // must be called here !!!
    }

    return buf;
}
```

Прочитайте Xamarin.Android - Bluetooth-связь онлайн: https://riptutorial.com/ru/xamarin-android/topic/10844/xamarin-android---bluetooth-связь

# глава 4: Xamarin.Android - Как создать панель инструментов

## замечания

Дорогая команда,

Я думаю, что неплохо было бы упомянуть официальную документацию на Android, где подробно описано управление панелью:

https://developer.android.com/reference/android/support/v7/widget/Toolbar.html

Также есть интересный контент о библиотеке Android.Support.v7, используемой в примере:

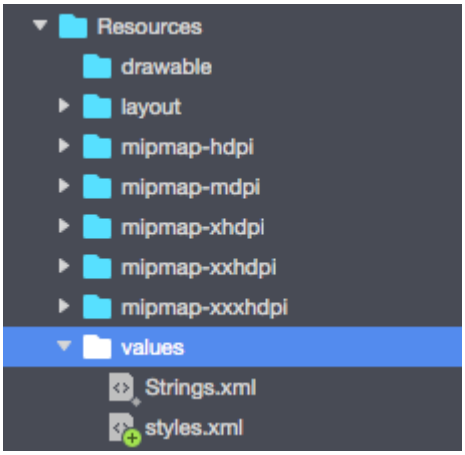
https://developer.android.com/training/appbar/index.html

## Examples

### Добавить панель инструментов в приложение Xamarin.Android

Во-первых, вам нужно добавить библиотеку Xamarin.Android.Support.V7.AppCompat для NuGet: https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat/

В папке «values» в разделе «Ресурсы» добавьте новый xml-файл под названием « styles.xml»:

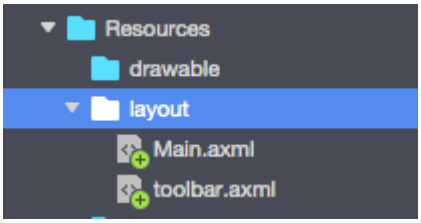



Файл styles.xml должен содержать код ниже:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
<style name="MyTheme" parent="MyTheme.Base">
</style>

<!-- Base theme applied no matter what API -->
<style name="MyTheme.Base" parent="Theme.AppCompat.Light.DarkActionBar">
```

```
<item name="windowNoTitle">true</item>
<!--We will be using the toolbar so no need to show ActionBar-->
<item name="windowActionBar">false</item>
<!-- Set theme colors from http://www.google.com/design/spec/style/color.html#color-color-
palette-->
<!-- colorPrimary is used for the default action bar background -->
<item name="colorPrimary">#2196F3</item>
<!-- colorPrimaryDark is used for the status bar -->
<item name="colorPrimaryDark">#1976D2</item>
<!-- colorAccent is used as the default value for colorControlActivated
     which is used to tint widgets -->
<item name="colorAccent">#FF4081</item>

<item name="colorControlHighlight">#FF4081</item>
<!-- You can also set colorControlNormal, colorControlActivated
     colorControlHighlight and colorSwitchThumbNormal. -->
```

Следующий шаг - добавить файл «toolbar.axml», содержащий определение элемента управления панели инструментов в папку «layout»:




Добавьте ниже код для определения панели инструментов:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:id="@+id/toolbar"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:minHeight="?attr/actionBarSize"
android:background="?attr/colorPrimary"
android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Теперь, пожалуйста, откройте файл «Main.axml» и добавьте ниже код чуть ниже закрывающего тега для первого макета. Ваш код должен выглядеть следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">

    <include android:id="@+id/toolbar" layout="@layout/toolbar" />

</LinearLayout>
```

Теперь вам нужно добавить информацию о теме, которую использует ваше приложение. Откройте файл «AndroidManifest» и добавьте информацию о теме в тег «application»:

```
<application android:theme="@style/MyTheme" android:allowBackup="true"
android:icon="@mipmap/icon" android:label="@string/app_name">
```

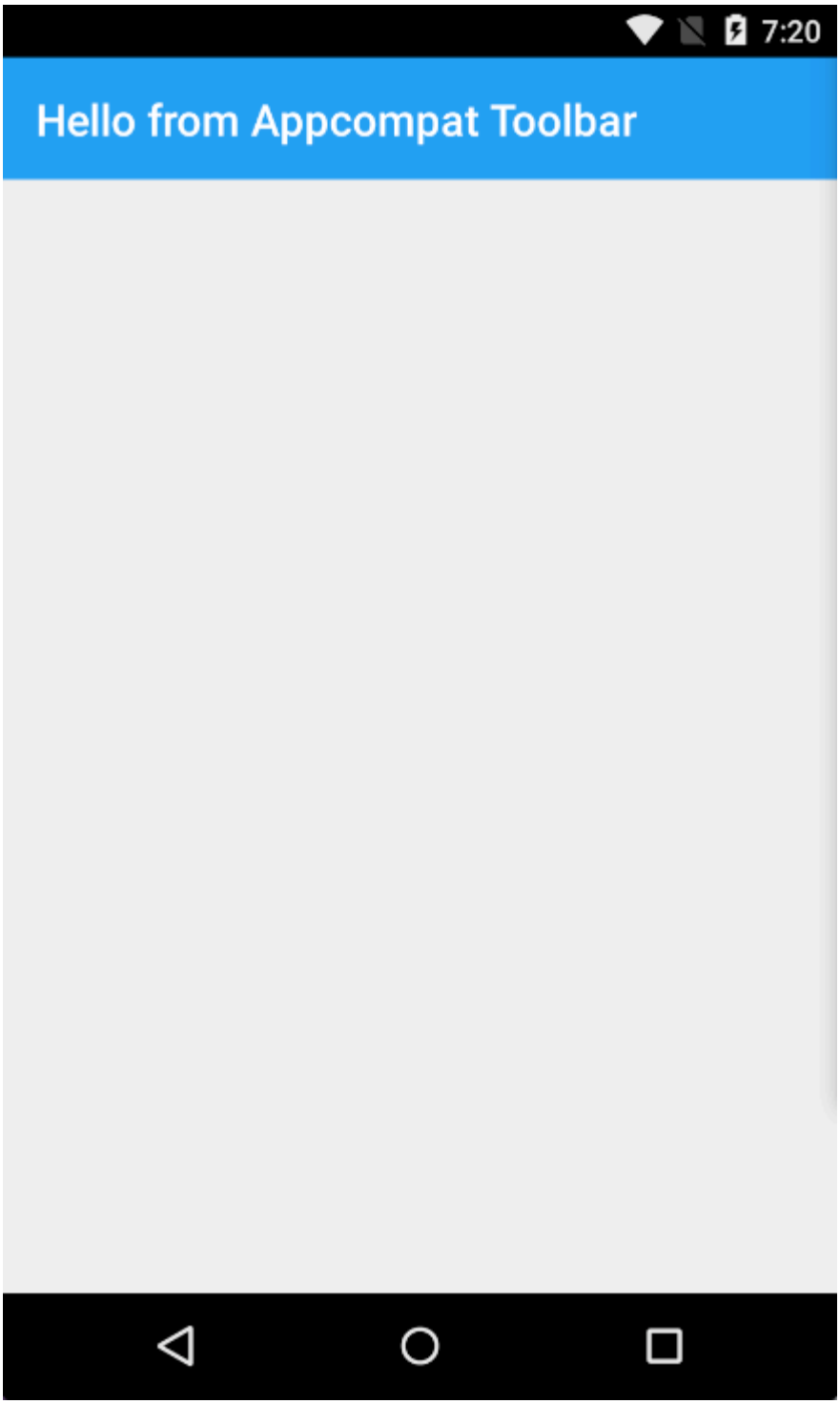
Последний шаг - подключить панель инструментов в файле Activity. Откройте файл « MainActivity.cs». Вы должны изменить вывод из «Активность» в «AppCompatActivity». Теперь получите ссылку на панель инструментов и установите ее как панель инструментов по умолчанию для активности в методе «OnCreate». Вы также можете определить название:

```
var toolbar = FindViewById<Android.Support.V7.Widget.Toolbar>(Resource.Id.toolbar);
        SetSupportActionBar(toolbar);
        SupportActionBar.Title = "Hello from Appcompat Toolbar";
```

Целый метод должен выглядеть следующим образом:

```
protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        SetContentView(Resource.Layout.Main);

        var toolbar = FindViewById<Android.Support.V7.Widget.Toolbar>(Resource.Id.toolbar);
        SetSupportActionBar(toolbar);
        SupportActionBar.Title = "Hello from Appcompat Toolbar";
    }
```

Перестройте проект и запустите его, чтобы увидеть результат:




Прочитайте Xamarin.Android - Как создать панель инструментов онлайн: https://riptutorial.com/ru/xamarin-android/topic/4755/xamarin-android---как-создать-панель-инструментов

# глава 5: Диалоги

## замечания

### Настройка `Context` диалога

При создании `Dialog` из `Activiy` мы можем использовать `this` как контекст.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

С помощью `Fragments` мы используем свойство `Context` .

```
AlertDialog.Builder builder = new AlertDialog.Builder(Context);
```

---

### Типы кнопок

`SetNeutralButton()` может использоваться для простого уведомления и подтверждения того, что уведомление читается. `SetPositiveButton()` может использоваться для подтверждения, например: «Вы действительно хотите удалить этот элемент?» `SetNegativeButton()` предназначен для отклонения диалога и отмены его действия.

---

### Отключить отмену от подноса

Если мы хотим убедиться, что пользователь не может `SetCanceable(false)` диалог с помощью кнопки «Назад», мы можем вызвать `SetCanceable(false)` . Это работает только для кнопки «Назад».

---

### вращение

Если экран повернут, а диалог виден, он будет отклонен, и действия ok и cancel не будут вызываться. Вам нужно будет обработать это внутри своей деятельности и повторно показать диалоговое окно после того, как активность будет перезагружена.

Чтобы обойти это, вместо этого используйте `DialogFragment` .

## Examples

### Диалоговое окно оповещения

Создание диалогового окна оповещения

```
AlertDialog.Builder builder = new AlertDialog.Builder(Context);
```

```
builder.SetIcon(Resource.Drawable.Icon);
builder.SetTitle(title);
builder.SetMessage(message);

builder.SetNeutralButton("Neutral", (evt, args) => {
    // code here for handling the Neutral tap
});

builder.SetPositiveButton("Ok", (evt, args) => {
    // code here for handling the OK tap
});

builder.SetNegativeButton("Cancel", (evt, args) => {
    // code here for handling the Cancel tap
});

builder.SetCancelable(false);
builder.Show();
```

Прочитайте Диалоги онлайн: https://riptutorial.com/ru/xamarin-android/topic/2510/диалоги

# глава 6: Диалоги

## параметры

| общедоступный общедоступный метод | использование |
|---|---|
| SetTitle (String) | Устанавливает заголовок для диалога |
| SetIcon (Drawable) | Значок настройки для диалогового окна предупреждения |
| SetMessage (строка) | Установите сообщение для отображения. |
| SetNegativeButton (String, EventHandler) | Установите прослушиватель, который будет вызываться, когда нажата отрицательная кнопка диалога. |
| SetPositiveButton (String, EventHandler) | Установите прослушиватель, который будет вызываться, когда нажата положительная кнопка диалога. |
| SetNeutralButton (String, EventHandler) | Установите прослушиватель, который будет вызываться, когда нажата нейтральная кнопка диалога. |
| SetOnCancelListener (IDialogInterfaceOnCancelListener) | Устанавливает обратный вызов, который будет вызываться, если диалог отменен. |
| SetOnDismissListener (IDialogInterfaceOnDismissListener) | Устанавливает обратный вызов, который вызывается, когда диалог отклоняется по какой-либо причине. |
| Шоу() | Создает AlertDialog с аргументами, предоставленными этому строителю, и диалоговым окном Dialog.Show. |

## замечания

**Требования**

Пространство имен: Android.App

Сборка: Mono.Android (в Mono.Android.dll)

Монтажные версии: 0.0.0.0

---

**Общественные конструкторы**

AlertDialog.Builder (контекст): -

Конструктор использует контекст для этого строителя и созданный AlertDialog.

AlertDialog.Builder (Context, Int32): -

Конструктор использует контекст и тему для этого строителя и созданный AlertDialog.

---

**Использование Material Design AlertDialog**

Чтобы использовать современный AlertDialog:

1. Установить библиотеку поддержки V7 AppCompat из пакетов NuGet
2. Замените AlertDialog на Android.Support.V7.App.AlertDialog или добавьте следующую инструкцию вверху, чтобы сделать диалог блистающим.

```
        using AlertDialog = Android.Support.V7.App.AlertDialog;
```

# Examples

## AlertDialog

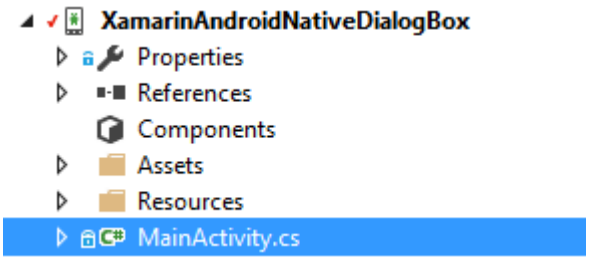
```
// 1. Instantiate an AlertDialog.Builder with its constructor
// the parameter this is the context (usually your activity)
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// 2. Chain together various setter methods to set the dialog characteristics
builder.SetMessage(Resource.String.dialog_message)
       .SetTitle(Resource.String.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.Create();

dialog.Show();
```

## Пример простого предупреждения

Мы создадим простой диалог оповещений в Xamarin.Android

Теперь, учитывая, что вы прошли руководство по началу работы из документации.

У вас должна быть структура проекта следующим образом:




Ваша основная деятельность должна выглядеть так:

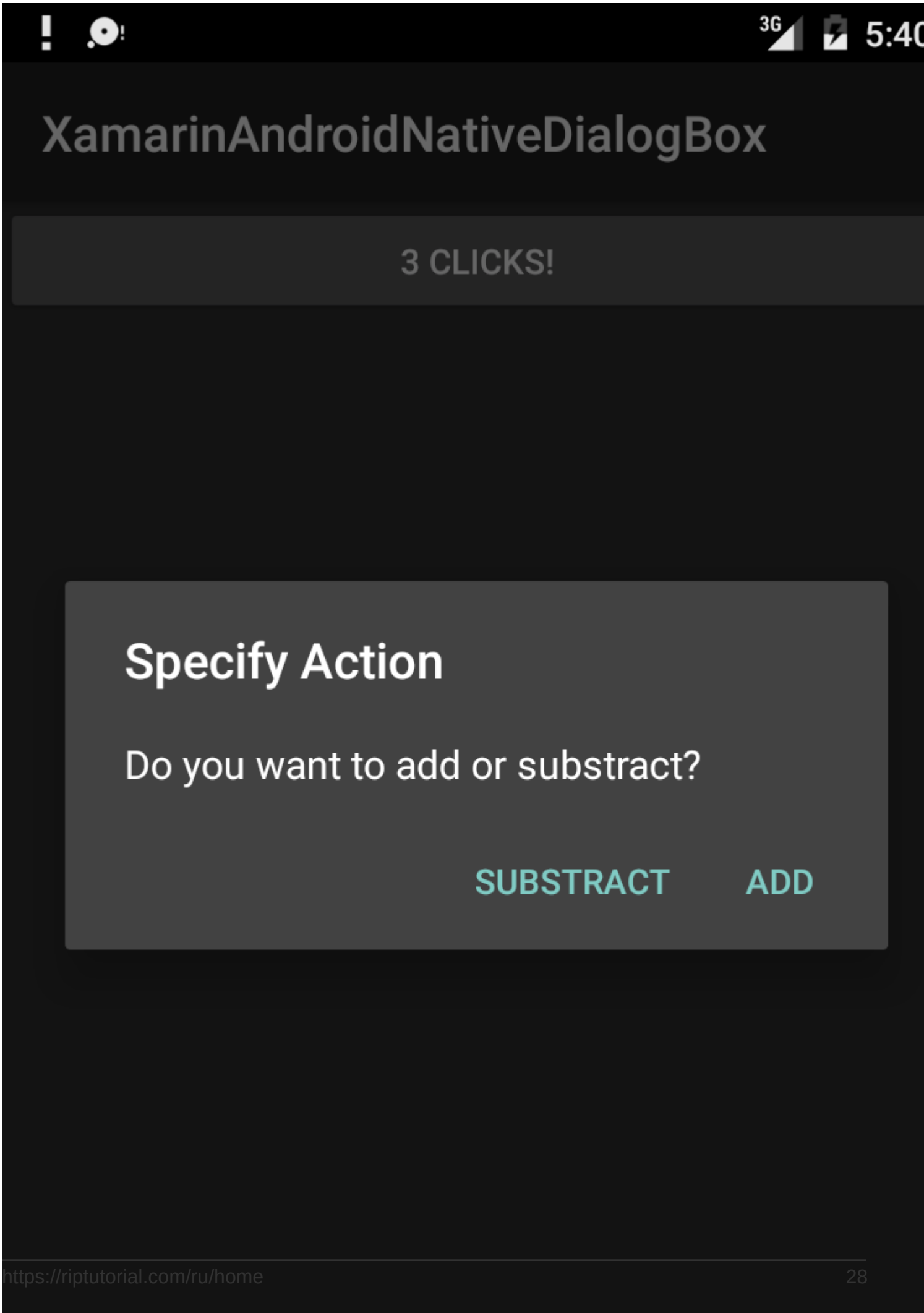
```
public class MainActivity : Activity
{
int count = 1;

protected override void OnCreate(Bundle bundle)
{
base.OnCreate(bundle);

// Set our view from the "main" layout resource
SetContentView(Resource.Layout.Main);

// Get our button from the layout resource,
// and attach an event to it
Button button = FindViewById<Button>(Resource.Id.MyButton);

button.Click += delegate { button.Text = string.Format("{0} clicks!", count++); };
}
}
```

Теперь, что мы сделаем, вместо того, чтобы добавлять один к счетчику при нажатии кнопки, мы спросим пользователя, хочет ли он добавить или вычесть его в простом диалоговом окне оповещений

И нажатие кнопки «Положительно» или «минус» мы предпримем действие.

```
button.Click += delegate {
AlertDialog.Builder alert = new AlertDialog.Builder(this);
alert.SetTitle("Specify Action");
alert.SetMessage("Do you want to add or substract?");

alert.SetPositiveButton("Add", (senderAlert, args) =>
{
count++;
button.Text = string.Format("{0} clicks!", count);
 });

 alert.SetNegativeButton("Substract", (senderAlert, args) =>
 {
 count--;
 button.Text = string.Format("{0} clicks!", count);
 });

 Dialog dialog = alert.Create();
     dialog.Show();
};
```

Скриншот:


XamarinAndroidNativeDialogBox
3 CLICKS!
Specify Action
Do you want to add or substract?
SUBSTRACT
ADD

# глава 7: Жизненный цикл приложения - Xamarin.Andorid

## Вступление

Жизненный цикл приложения Xamarin.Android совпадает с обычным Android-приложением. Говоря о жизненном цикле, нам нужно поговорить о: жизненном цикле приложения, жизненном цикле активности и жизненном цикле фрагментов.

Ниже я попытаюсь дать хорошее описание и способ их использования. Я получил эту документацию из официальной документации Android и Xamarin и многих полезных веб-ресурсов, приведенных ниже в разделе замечаний.

## замечания

Интересные ссылки для широкого ознакомления с жизненным циклом приложений Android:

https://developer.android.com/reference/android/app/Activity.html

http://www.vogella.com/tutorials/AndroidLifeCycle/article.html

https://github.com/xxv/android-lifecycle

https://developer.android.com/guide/components/fragments.html

https://developer.xamarin.com/guides/android/platform_features/fragments/part_1_-_creating_a_fragment/

https://developer.android.com/guide/components/activities/activity-lifecycle.html

## Examples

### Жизненный цикл приложения

Прежде всего, вы должны знать, что вы можете расширить класс Android.Application, чтобы вы могли получить доступ к двум важным методам, связанным с жизненным циклом приложения:

- OnCreate - вызывается, когда приложение запускается, до создания каких-либо других объектов приложения (например, MainActivity).
- OnTerminate - этот метод предназначен для использования в эмулируемых средах процесса. Он никогда не будет вызываться на производственном устройстве Android,

где процессы удаляются, просто убивая их; При этом не выполняется код пользователя (включая этот обратный вызов). Из документации: https://developer.android.com/reference/android/app/Application.html#onTerminate ()

В приложении Xamarin.Android вы можете расширить класс Application так, как показано ниже. Добавьте в проект новый класс под названием «MyApplication.cs»:

```
[Application]
public class MyApplication : Application
{
    public MyApplication(IntPtr handle, JniHandleOwnership ownerShip) : base(handle,
ownerShip)
    {
    }

    public override void OnCreate()
    {
        base.OnCreate();
    }

    public override void OnTerminate()
    {
        base.OnTerminate();
    }
}
```

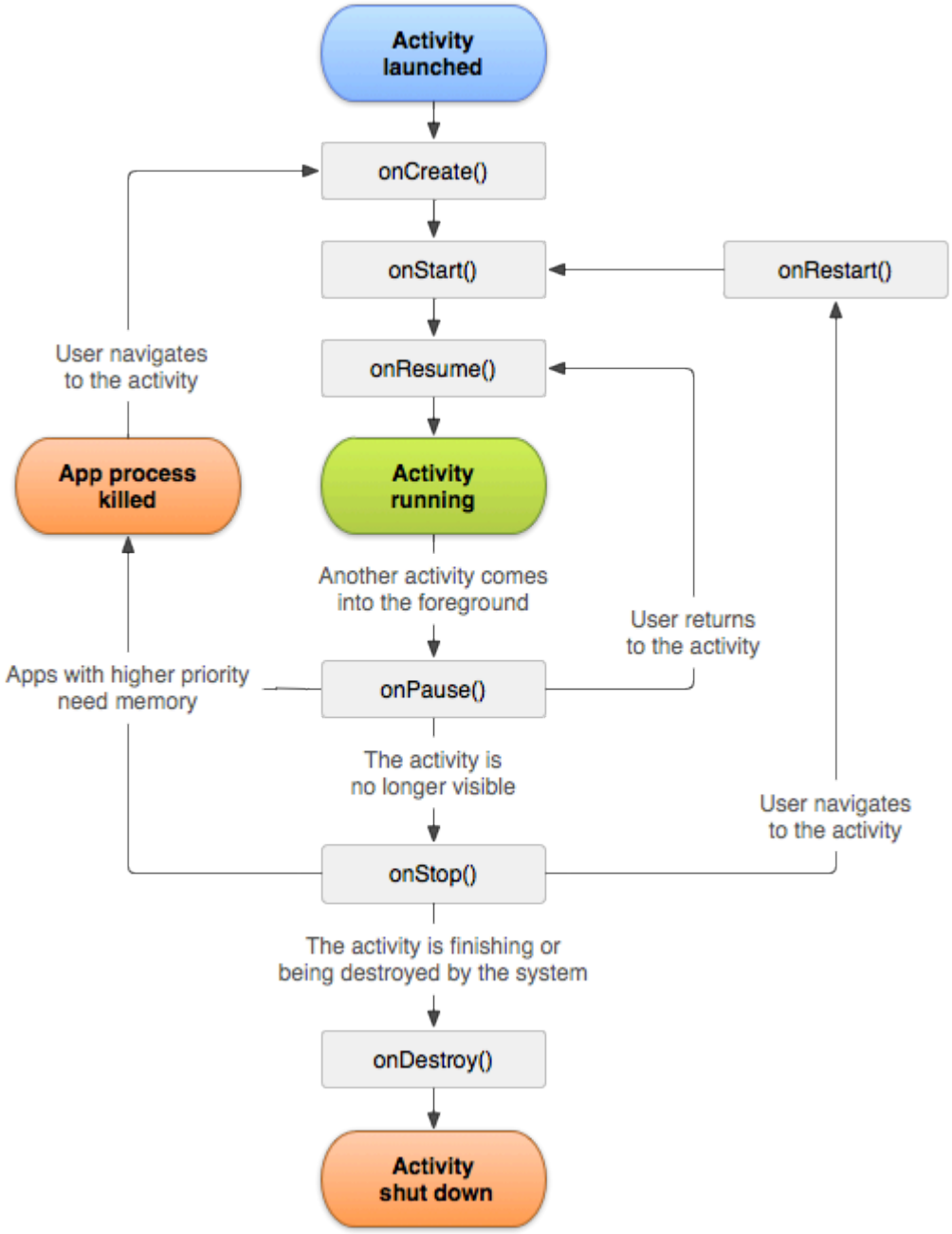
Как вы писали выше, вы можете использовать метод OnCreate. Вы можете, например, инициализировать локальную базу данных здесь или настроить некоторую дополнительную конфигурацию.

Существует также множество методов, которые можно переопределить, например: OnConfigurationChanged или OnLowMemory.

## Жизненный цикл деятельности

Жизненный цикл деятельности довольно сложный. Как вы знаете, активность - это одна страница в приложении для Android, где пользователь может взаимодействовать с ней.

На приведенной ниже диаграмме вы можете увидеть, как выглядит жизненный цикл Android Activity:




Как вы видите, существует определенный поток жизненного цикла Activity. В мобильном приложении у вас есть, конечно, методы в каждом классе Activity, которые обрабатывают определенный фрагмент жизненного цикла:

```
[Activity(Label = "LifecycleApp", MainLauncher = true, Icon = "@mipmap/icon")]
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        Log.Debug("OnCreate", "OnCreate called, Activity components are being created");

        // Set our view from the "main" layout resource
        SetContentView(Resource.Layout.MainActivity);
    }

    protected override void OnStart()
```

```
    {
        Log.Debug("OnStart", "OnStart called, App is Active");
        base.OnStart();
    }

    protected override void OnResume()
    {
        Log.Debug("OnResume", "OnResume called, app is ready to interact with the user");
        base.OnResume();
    }

    protected override void OnPause()
    {
        Log.Debug("OnPause", "OnPause called, App is moving to background");
        base.OnPause();
    }

    protected override void OnStop()
    {
        Log.Debug("OnStop", "OnStop called, App is in the background");
        base.OnStop();
    }

    protected override void OnDestroy()
    {
        base.OnDestroy();
        Log.Debug("OnDestroy", "OnDestroy called, App is Terminating");
    }
}
```

В официальной документации на Android есть хорошее описание:

- Весь жизненный цикл активности происходит между первым вызовом onCreate (Bundle) до одного окончательного вызова onDestroy (). Активность будет выполнять настройку «глобального» состояния в onCreate () и освободить все оставшиеся ресурсы в onDestroy (). Например, если у него есть поток, выполняющийся в фоновом режиме для загрузки данных из сети, он может создать этот поток в onCreate (), а затем остановить поток в onDestroy ().

- Видимое время жизни активности происходит между вызовом onStart () до соответствующего вызова onStop (). В течение этого времени пользователь может видеть активность на экране, хотя он не может быть на переднем плане и взаимодействовать с пользователем. Между этими двумя методами вы можете поддерживать ресурсы, необходимые для отображения активности пользователю. Например, вы можете зарегистрировать BroadcastReceiver в onStart (), чтобы отслеживать изменения, влияющие на ваш пользовательский интерфейс, и отменить регистрацию в onStop (), когда пользователь больше не видит, что вы показываете. Методы onStart () и onStop () можно вызывать несколько раз, так как активность становится видимой и скрытой для пользователя.

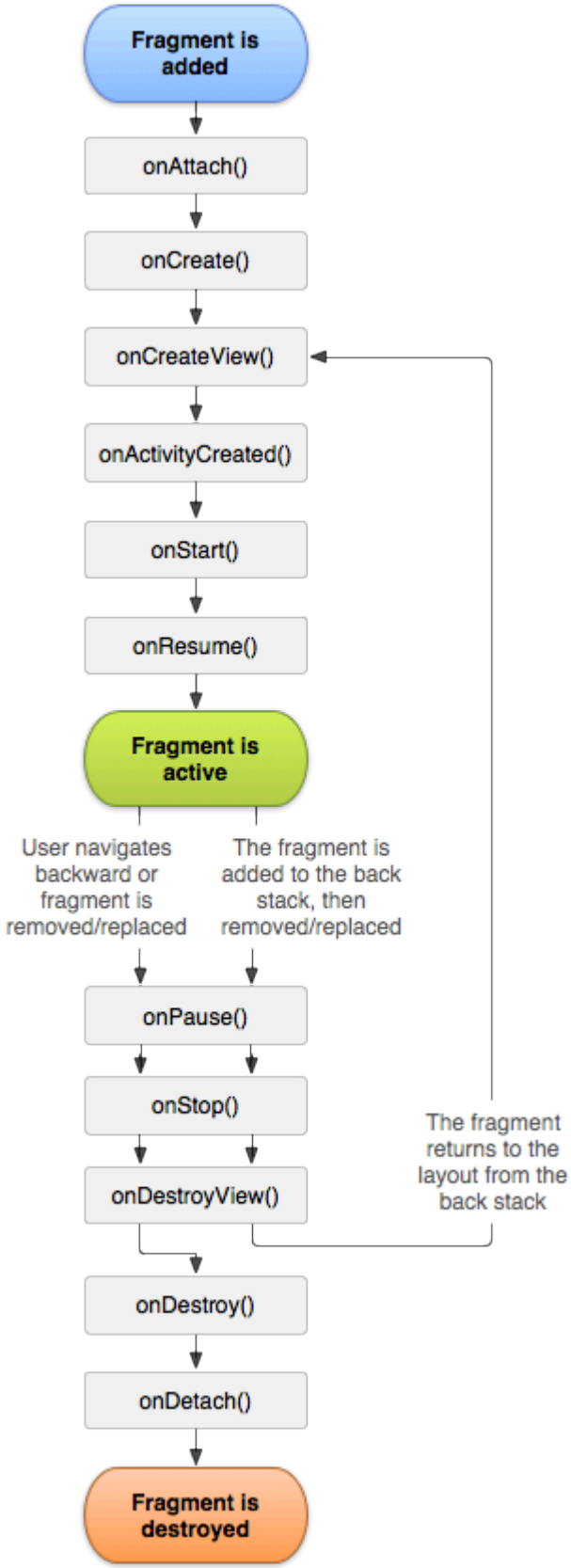
- Передний жизненный цикл активности происходит между вызовом onResume () до соответствующего вызова onPause (). За это время активность перед всеми другими

действиями и взаимодействием с пользователем. Активность может часто проходить между возобновленными и приостановленными состояниями - например, когда устройство переходит в спящий режим, когда доставляется результат деятельности, когда доставляется новое намерение, поэтому код в этих методах должен быть достаточно легким.

## Жизненный цикл фрагментов

Как вы знаете, у вас может быть одна активность, но в нее встроены разные фрагменты. Вот почему жизненный цикл фрагментов также важен для разработчиков.

На приведенной ниже диаграмме вы можете увидеть, как выглядит жизненный цикл фрагмента Android:




Как описано в официальной документации на Android, вы должны реализовать как минимум три метода:

- OnCreate - система вызывает это при создании фрагмента. В рамках вашей реализации вы должны инициализировать основные компоненты фрагмента, которые вы хотите сохранить, когда фрагмент приостановлен или остановлен, а затем

возобновлен.

- OnCreateView - система вызывает это, когда пришло время, чтобы фрагмент впервые ввел свой пользовательский интерфейс. Чтобы нарисовать пользовательский интерфейс для вашего фрагмента, вы должны вернуть представление из этого метода, который является корнем макета вашего фрагмента. Вы можете вернуть значение null, если фрагмент не предоставляет пользовательский интерфейс.
- OnPause - система вызывает этот метод как первое указание на то, что пользователь покидает фрагмент (хотя это не всегда означает уничтожение фрагмента). Обычно вы должны фиксировать любые изменения, которые должны сохраняться за пределами текущего сеанса пользователя (поскольку пользователь может не вернуться).

Вот пример реализации в Xamarin.Android:

```
public class MainFragment : Fragment
{
    public override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);

        // Create your fragment here
        // You should initialize essential components of the fragment
        // that you want to retain when the fragment is paused or stopped, then resumed.
    }

    public override View OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState)
    {
        // Use this to return your custom view for this Fragment
        // The system calls this when it's time for the fragment to draw its user interface
for the first time.

        var mainView = inflater.Inflate(Resource.Layout.MainFragment, container, false);
        return mainView;
    }

    public override void OnPause()
    {
        // The system calls this method as the first indication that the user is leaving the
fragment

        base.OnPause();
    }
}
```

Конечно, здесь вы можете добавить дополнительные методы, если хотите обрабатывать разные состояния.

### Полный образец GitHub

Если вы хотите получить базовый проект со способами, описанными ниже, вы можете

скачать шаблон приложения Xamarin.Android из моего GitHub. Вы можете найти примеры для:

- Методы жизненного цикла приложения
- Методы жизненного цикла
- Методы жизненного цикла фрагментов

https://github.com/Daniel-Krzyczkowski/XamarinAndroid/tree/master/AndroidLifecycle/LifecycleApp

Прочитайте Жизненный цикл приложения - Xamarin.Andorid онлайн: https://riptutorial.com/ru/xamarin-android/topic/8842/жизненный-цикл-приложения---xamarin-andorid

# глава 8: Как исправить ориентацию изображения, снятого с устройства Android

## замечания

1. Этот пример приложения доступен на моем GitHub ниже:

https://github.com/Daniel-Krzyczkowski/XamarinAndroid/tree/master/AndroidPictureOrientation/PictureOrientationApp

2. Документация компонентов Xamarin Mobile приведена ниже:

https://components.xamarin.com/view/xamarin.mobile

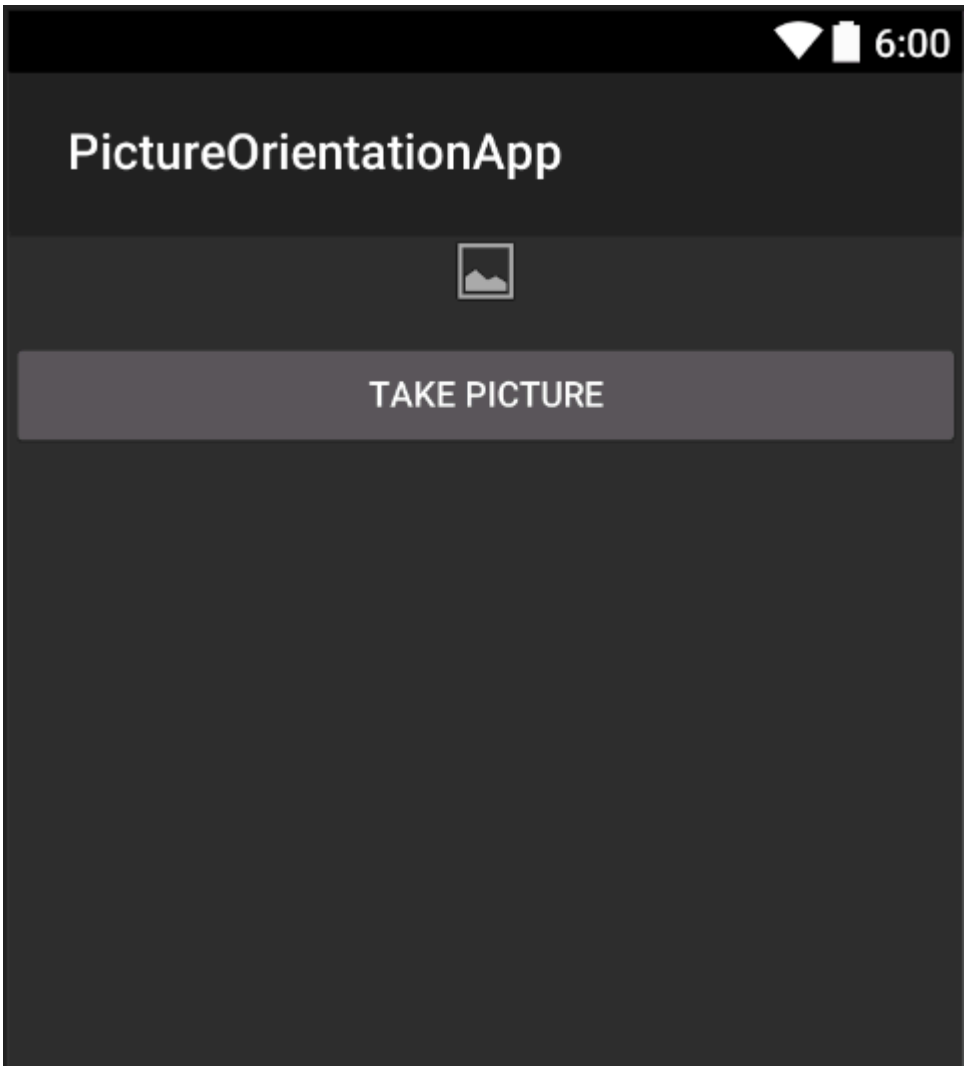
## Examples

### Как исправить ориентацию изображения, снятого с устройства Android

В этом примере показано, как правильно отображать изображение и отображать его на устройстве Android.

Во-первых, мы должны создать пример приложения с одной кнопкой и одним изображением. Как только пользователь нажимает на кнопку, камера запускается, и после того, как пользователь выбирает изображение, он будет отображаться с соответствующей ориентацией на экране.

1. Добавьте кнопку с именем «TakePictureButton» и изображение с именем « TakenPictureImageView»:

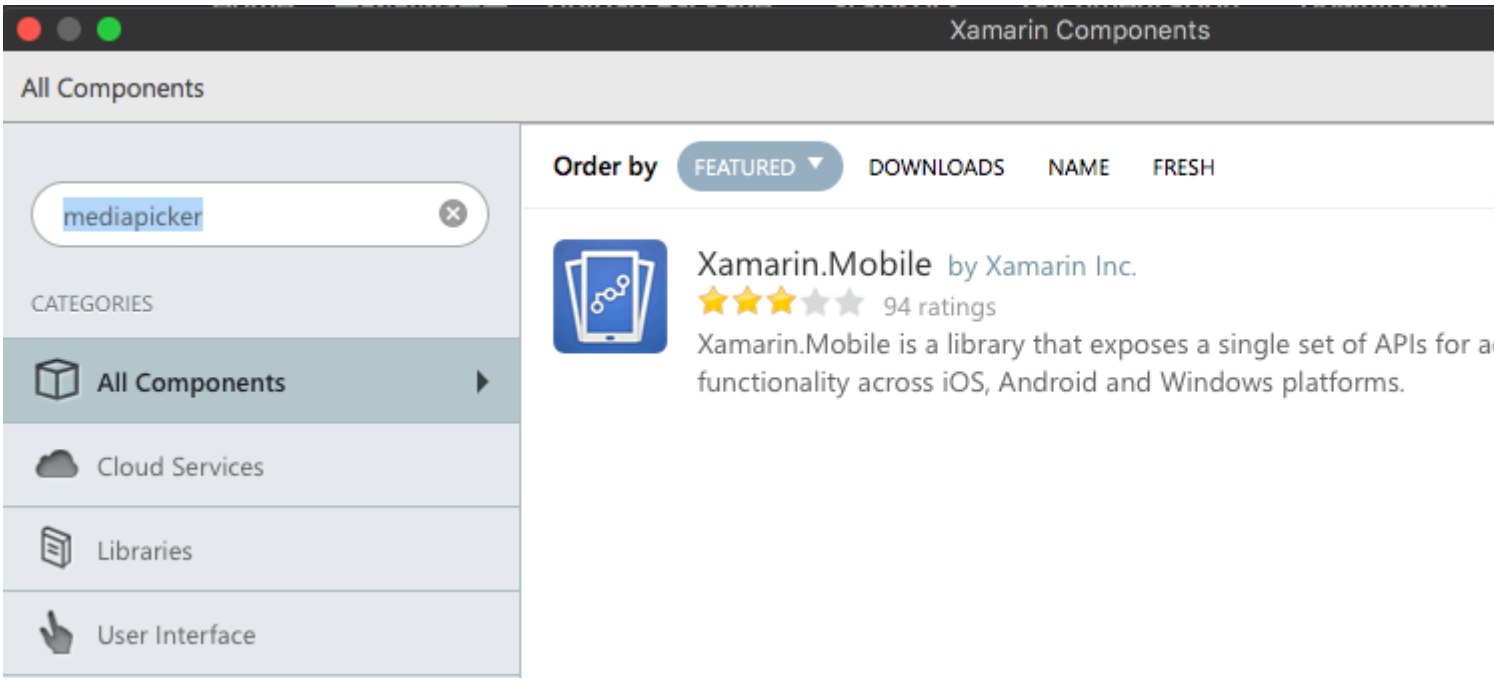



2. Теперь откройте код активности:

Сначала обратитесь к своим элементам управления:

```
ImageView _takenPictureImageView;
Button _takePictureButton;

    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        SetContentView(Resource.Layout.Main);

        _takenPictureImageView = FindViewById<ImageView>(Resource.Id.TakenPictureImageView);
        _takePictureButton = FindViewById<Button>(Resource.Id.TakePictureButton);

        _takePictureButton.Click += delegate
        {
            takePicture();
        };
    }
```

3. В нашей заявке мы будем использовать компонент Xamarin Mobile, доступный в магазине компонентов:




4. Как только вы добавите его в проект, мы можем двигаться дальше. Добавьте ниже код, который отвечает за запуск камеры. Этот метод следует вызвать нажатием кнопки, как вы можете видеть в приведенном выше коде:

```
void takePicture()
 {
     var picker = new MediaPicker(this);
     DateTime now = DateTime.Now;
     var intent = picker.GetTakePhotoUI(new StoreCameraMediaOptions
     {
         Name = "picture_" + now.Day + "_" + now.Month + "_" + now.Year + ".jpg",
         Directory = null
     });
     StartActivityForResult(intent, 1);
 }
```

5. Как только пользователь сделает снимок, мы должны отобразить его в правильной ориентации. Для этого используйте ниже метод. Он отвечает за извлечение exif-информации из принятого изображения (включая ориентацию в момент съемки) и создание растрового изображения с правильной ориентацией:

```
Bitmap loadAndResizeBitmap(string filePath)
 {
     BitmapFactory.Options options = new BitmapFactory.Options { InJustDecodeBounds =
true };
     BitmapFactory.DecodeFile(filePath, options);

     int REQUIRED_SIZE = 100;
     int width_tmp = options.OutWidth, height_tmp = options.OutHeight;
     int scale = 4;
     while (true)
     {
         if (width_tmp / 2 < REQUIRED_SIZE || height_tmp / 2 < REQUIRED_SIZE)
             break;
         width_tmp /= 2;
```

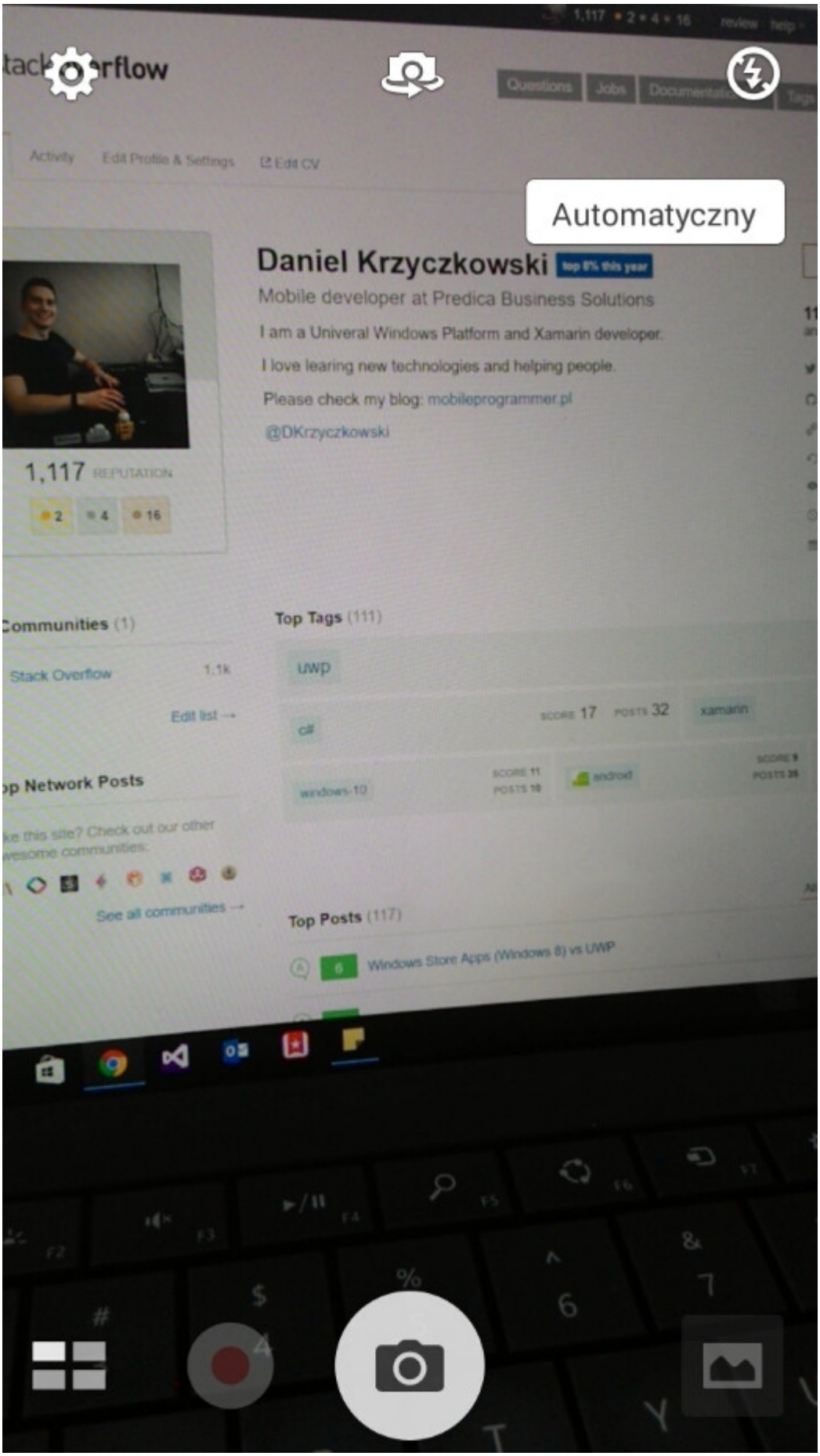
```
        height_tmp /= 2;
        scale++;
    }

    options.InSampleSize = scale;
    options.InJustDecodeBounds = false;
    Bitmap resizedBitmap = BitmapFactory.DecodeFile(filePath, options);

    ExifInterface exif = null;
    try
    {
        exif = new ExifInterface(filePath);
        string orientation = exif.GetAttribute(ExifInterface.TagOrientation);

        Matrix matrix = new Matrix();
        switch (orientation)
        {
            case "1": // landscape
                break;
            case "3":
                matrix.PreRotate(180);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "4":
                matrix.PreRotate(180);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "5":
                matrix.PreRotate(90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "6": // portrait
                matrix.PreRotate(90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "7":
                matrix.PreRotate(-90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "8":
                matrix.PreRotate(-90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
```

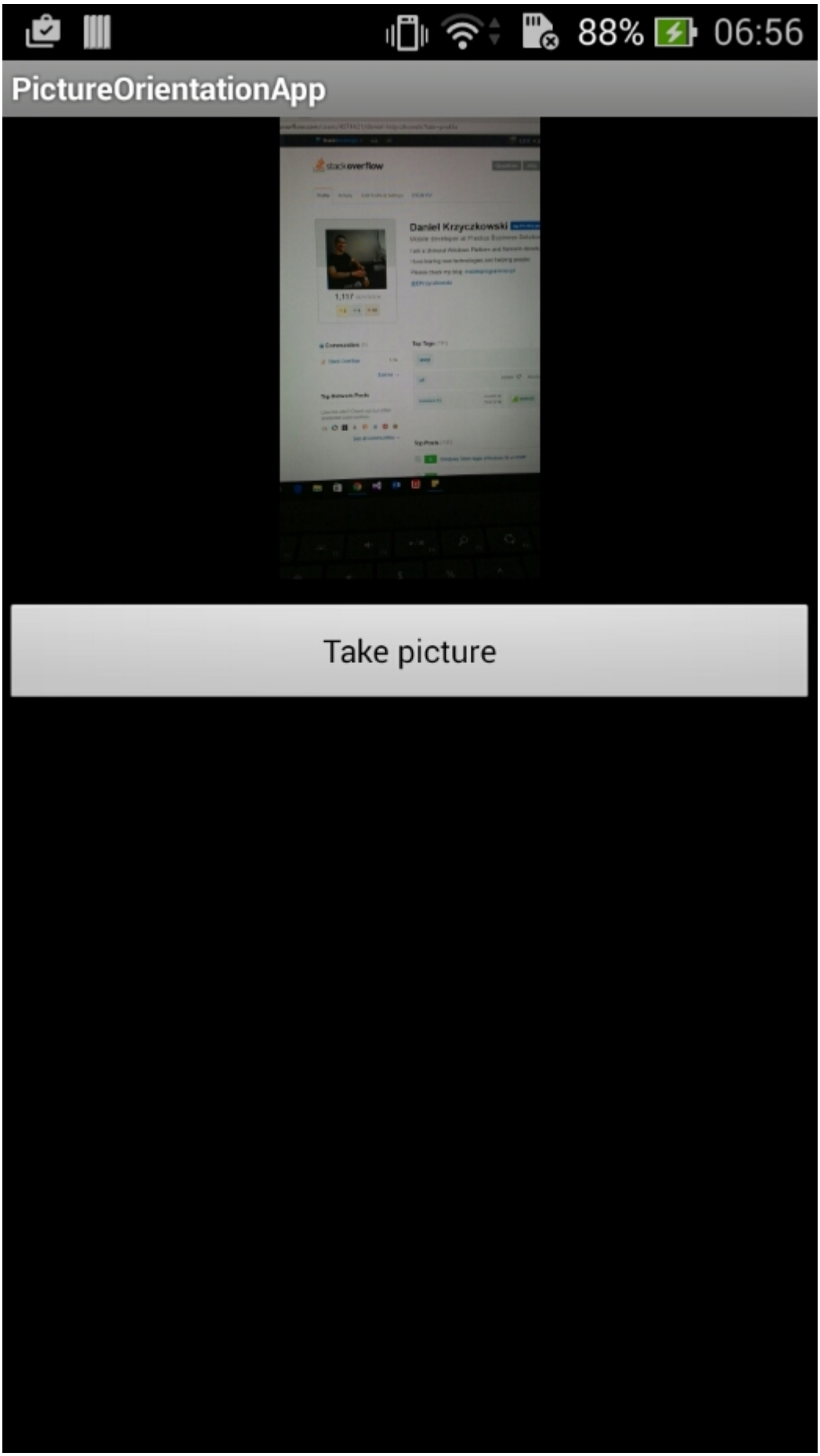
```
        }

        return resizedBitmap;
    }

    catch (IOException ex)
    {
        Console.WriteLine("An exception was thrown when reading exif from media
file...:" + ex.Message);
        return null;
    }
 }
```

6. Вышеуказанный метод следует вызывать в методе OnActivityResult, который вызывается после того, как пользователь делает снимок:

```
protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
 {
    base.OnActivityResult(requestCode, resultCode, data);

    if (requestCode == 1)
    {
        if (resultCode == Result.Ok)
        {
            data.GetMediaFileExtraAsync(this).ContinueWith(t =>
            {
                using (Bitmap bmp = loadAndResizeBitmap(t.Result.Path))
                {
                    if (bmp != null)
                    _takenPictureImageView.SetImageBitmap(bmp);
                }

            }, TaskScheduler.FromCurrentSynchronizationContext());
        }
    }
 }
```

7. Запустите приложение. Сделайте снимок, чтобы увидеть результат:


Automatyczny
Daniel Krzyczkowski
Mobile developer at Predica Business Solutions
I am a Univeral Windows Platform and Xamarin developer.
I love learing new technologies and helping people.
Please check my blog: mobileprogrammer.pl
@DKrzyczkowski
1,117 REPUTATION
Communities (1)
Stack Overflow
Edit list
Top Tags (111)
uwp
c#
xamarin
android
Top Posts (117)
Windows Store Apps (Windows 8) vs UWP




Вот и все. Теперь у вас будет все, что вы сделали, чтобы изображение отображалось в правильной ориентации.

Прочитайте Как исправить ориентацию изображения, снятого с устройства Android онлайн: https://riptutorial.com/ru/xamarin-android/topic/6683/как-исправить-ориентацию-изображения--снятого-с-устройства-android

# глава 9: Наручники

## Examples

### Удаление типов

Можно дать указание генератору привязок Xamarin.Android игнорировать тип Java и не связывать его. Это делается путем добавления XML-элемента `remove-node` в файл metadata.xml:

```
<remove-node path="/api/package[@name='{package_name}']/class[@name='{name}']" />
```

### Внедрение интерфейсов Java

Если в java-библиотеке содержатся интерфейсы, которые должны быть реализованы пользователем (например, нажмите на прослушиватели, такие как `View.IOnClickListener` или обратные вызовы), класс реализации должен наследовать - прямо или косвенно - из `Java.Lang.Object` или `Java.Lang.Throwable` , Это обычная ошибка, так как этапы упаковки просто печатают предупреждение, которое легко пропускается:

> Тип «MyListener» реализует Android.Runtime.IJavaObject, но не наследует Java.Lang.Object. Он не поддерживается.

**Неправильно**

Использование этой реализации приведет к неожиданному поведению.

```
class MyListener : View.IOnClickListener
{
    public IntPtr Handle { get; }

    public void Dispose()
    {
    }


    public void OnClick(View v)
    {
        // ...
    }
}
```

**Правильный**

```
class MyListener :
    Java.Lang.Object, // this is the important part
    View.IOnClickListener
{
```

```
    public void OnClick(View v)
    {
        // ...
    }
}
```

### Библиотеки привязок могут переименовывать методы и интерфейсы

Не все в библиотеке привязок будет иметь такое же имя в C #, как и в Java.

В C # имена интерфейсов начинаются с «I», но Java не имеет такого соглашения. Когда вы импортируете библиотеку Java, интерфейс с именем `SomeInterface` станет `ISomeInterface` .

Аналогично, у Java нет таких свойств, как C #. Когда библиотека привязана, методы getter и setter могут быть реорганизованы как свойства. Например, следующий код Java

```
public int getX() { return someInt; }

public int setX(int someInt) { this.someInt = someInt; }
```

могут быть реорганизованы как

```
public int X { get; set; }
```

когда он связан.

Прочитайте Наручники онлайн: https://riptutorial.com/ru/xamarin-android/topic/771/наручники

# глава 10: Пользовательский ListView

## Examples

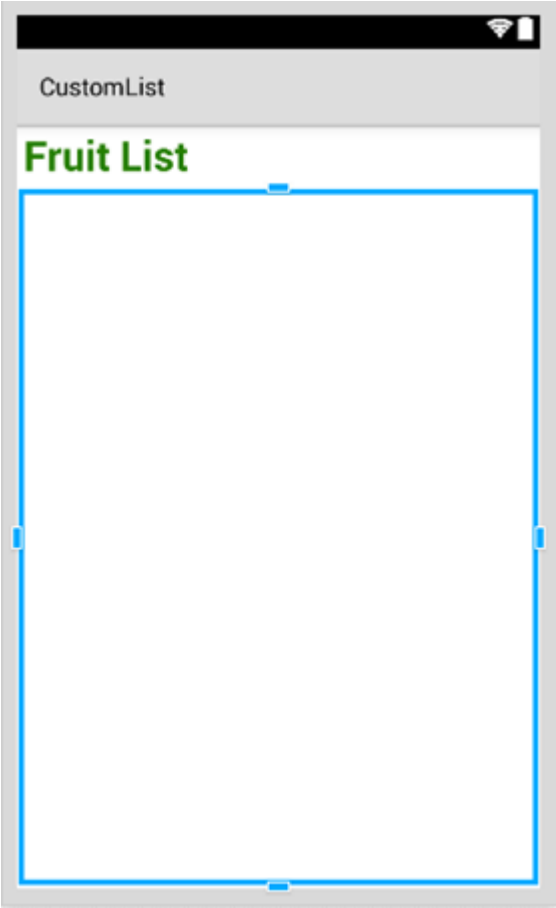
**Пользовательский список Listview состоит из строк, которые разработаны в соответствии с потребностями пользователей.**


Для макета над вашим файлом customrow.axml показано ниже

```
    <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="8dp">
    <ImageView
        android:id="@+id/Image"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:layout_alignParentLeft="true"
        android:layout_marginRight="8dp"
        android:src="@drawable/icon" />
    <TextView
        android:id="@+id/Text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@id/Image"
        android:layout_toRightOf="@id/Image"
        android:layout_marginTop="5dip"
        android:text="This is Line1"
        android:textSize="20dip"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/Text2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/Text1"
```

```
        android:layout_marginTop="1dip"
        android:text="This is line2"
        android:textSize="15dip"
        android:layout_toRightOf="@id/Image" />
</RelativeLayout>
```

Затем вы можете создать свой main.axml, который содержит текстовое представление для заголовка и списка.




Надеюсь, что это легко ...

Затем создайте свой класс Data.cs, который будет представлять ваши объекты строк

```
public class Data
{
    public string Heading;
    public string SubHeading;
    public string ImageURI;

    public Data ()
    {
        Heading = "";
        SubHeading = "";
        ImageURI = "";
    }
}
```

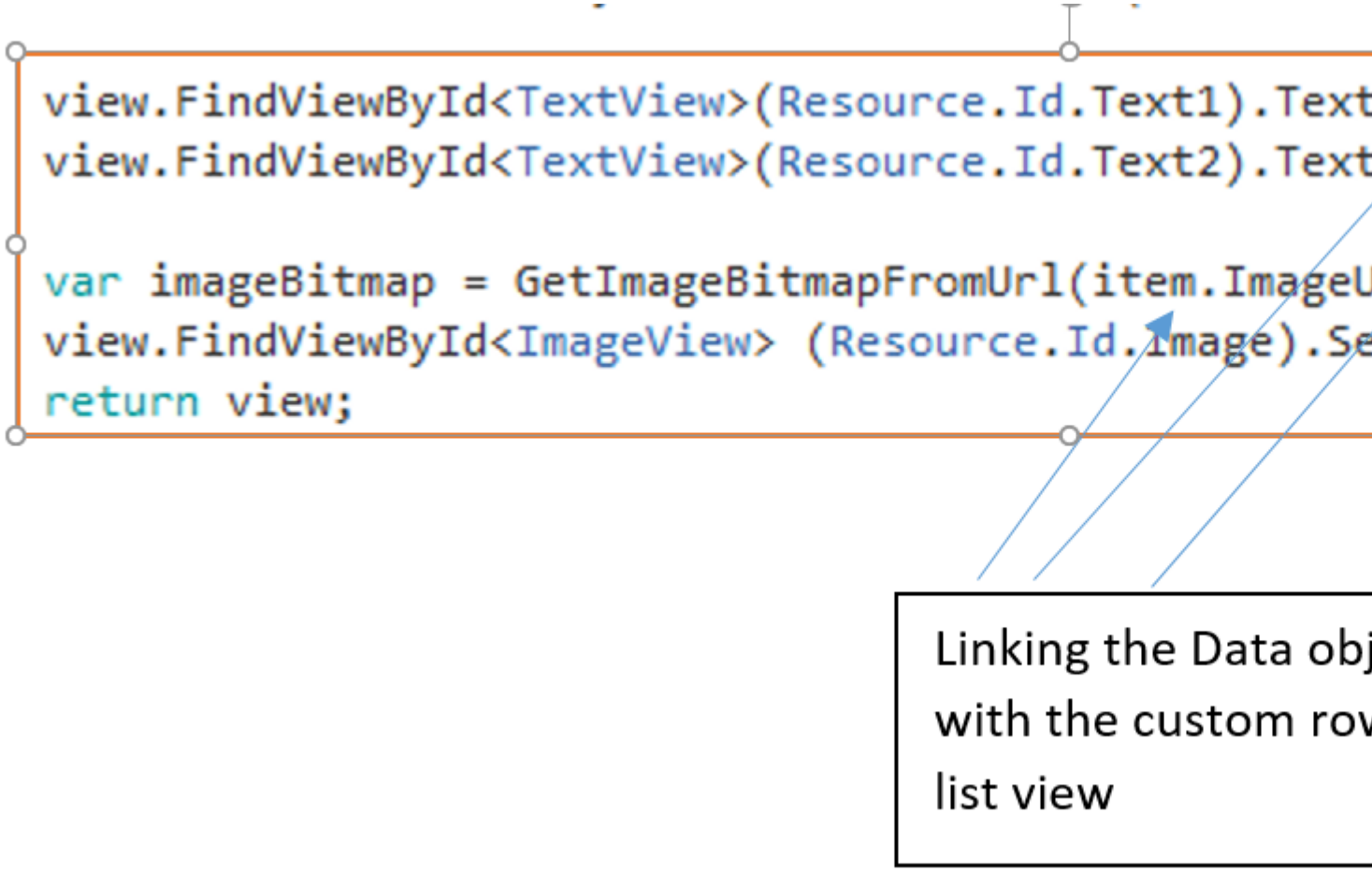
Далее вам понадобится класс DataAdapter.cs, адаптеры свяжут ваши данные с основным

видом

```
public class DataAdapter : BaseAdapter<Data> {

    List<Data> items;

    Activity context;
    public DataAdapter(Activity context, List<Data> items)
        : base()
    {
        this.context = context;
        this.items = items;
    }
    public override long GetItemId(int position)
    {
        return position;
    }
    public override Data this[int position]
    {
        get { return items[position]; }
    }
    public override int Count
    {
        get { return items.Count; }
    }
    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        var item = items[position];
        View view = convertView;
        if (view == null) // no view to re-use, create new
            view = context.LayoutInflater.Inflate(Resource.Layout.CustomRow, null);

        view.FindViewById<TextView>(Resource.Id.Text1).Text = item.Heading;
        view.FindViewById<TextView>(Resource.Id.Text2).Text = item.SubHeading;

        var imageBitmap = GetImageBitmapFromUrl(item.ImageURI);
        view.FindViewById<ImageView> (Resource.Id.Image).SetImageBitmap (imageBitmap);
        return view;
    }

    private Bitmap GetImageBitmapFromUrl(string url)
    {
        Bitmap imageBitmap = null;
        if(!(url=="null"))
            using (var webClient = new WebClient())
            {
                var imageBytes = webClient.DownloadData(url);
                if (imageBytes != null && imageBytes.Length > 0)
                {
                    imageBitmap = BitmapFactory.DecodeByteArray(imageBytes, 0,
imageBytes.Length);
                }
            }

        return imageBitmap;
    }

}
```

Самая важная часть - внутри функции GetView, здесь вы связываете свой объект со своей

пользовательской строкой.



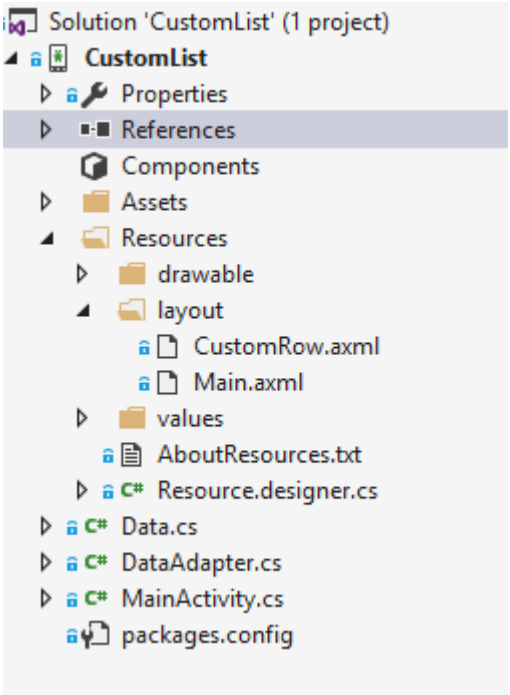

GetImageBitmapFromUrl не является частью адаптера данных, но я поставил его здесь для простоты.

Наконец мы приходим к MainActivity.cs

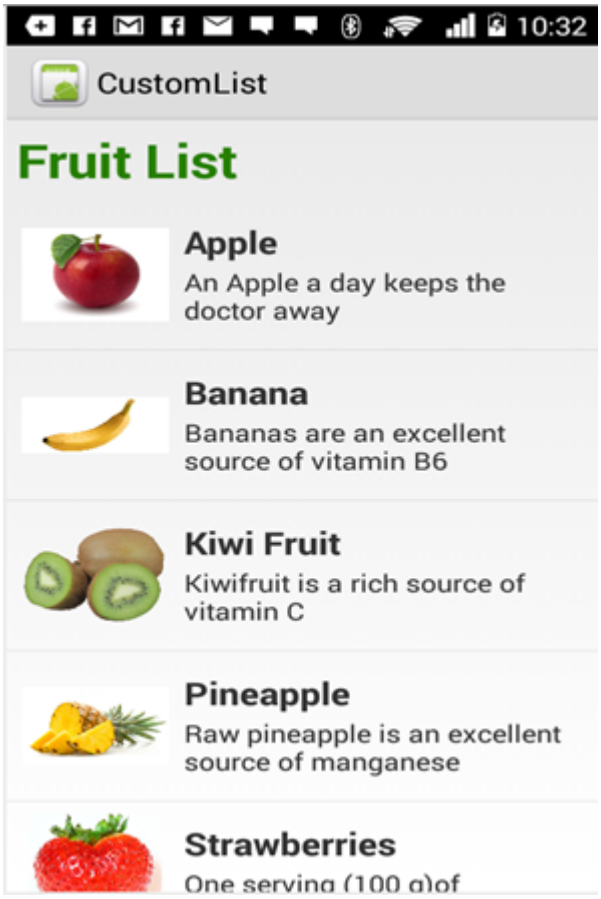
```
public class MainActivity : Activity
{

    ListView listView;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        // Set our view from the "main" layout resource
        SetContentView (Resource.Layout.Main);
        listView = FindViewById<ListView>(Resource.Id.List);

        List<Data> myList = new List<Data> ();

        Data obj = new Data ();
        obj.Heading = "Apple";
        obj.SubHeading = "An Apple a day keeps the doctor away";
        obj.ImageURI =
"http://www.thestar.com/content/dam/thestar/opinion/editorials/star_s_view_/2011/10/12/an_apple_a_day_
```

```
        myList.Add (obj);

        Data obj1 = new Data();
        obj1.Heading = "Banana";
        obj1.SubHeading = "Bananas are an excellent source of vitamin B6 ";
        obj1.ImageURI =
"http://www.bbcgoodfood.com/sites/bbcgoodfood.com/files/glossary/banana-crop.jpg";

        myList.Add(obj1);

        Data obj2 = new Data();
        obj2.Heading = "Kiwi Fruit";
        obj2.SubHeading = "Kiwifruit is a rich source of vitamin C";
        obj2.ImageURI = "http://www.wiffens.com/wp-content/uploads/kiwi.png";

        myList.Add(obj2);

        Data obj3 = new Data();
        obj3.Heading = "Pineapple";
        obj3.SubHeading = "Raw pineapple is an excellent source of manganese";
        obj3.ImageURI =
"http://www.medicalnewstoday.com/images/articles/276/276903/pineapple.jpg";

        myList.Add(obj3);

        Data obj4 = new Data();
        obj4.Heading = "Strawberries";
        obj4.SubHeading = "One serving (100 g)of strawberries contains approximately 33
kilocalories";
        obj4.ImageURI = "https://ecs3.tokopedia.net/newimg/product-
1/2014/8/18/5088/5088_8dac78de-2694-11e4-8c99-6be54908a8c2.jpg";

        myList.Add (obj4);
        listView.Adapter = new DataAdapter(this,myList);

    }
```

Ваша окончательная структура проекта показана ниже.



Если все в порядке, вы должны увидеть выход, как показано




Прочитайте Пользовательский ListView онлайн: https://riptutorial.com/ru/xamarin-android/topic/6406/пользовательский-listview

# глава 11: Публикация вашего Xamarin.Android APK

## Вступление

В этом разделе представлена информация о том, как подготовить приложение Xamarin.Android для режима выпуска и как его оптимизировать.

## Examples

### Подготовка APK в Visual Studio

Вы закончили свое приложение, протестировали его в режиме отладки и отлично работали. Теперь вы хотите подготовить его для публикации в Google Play Store.

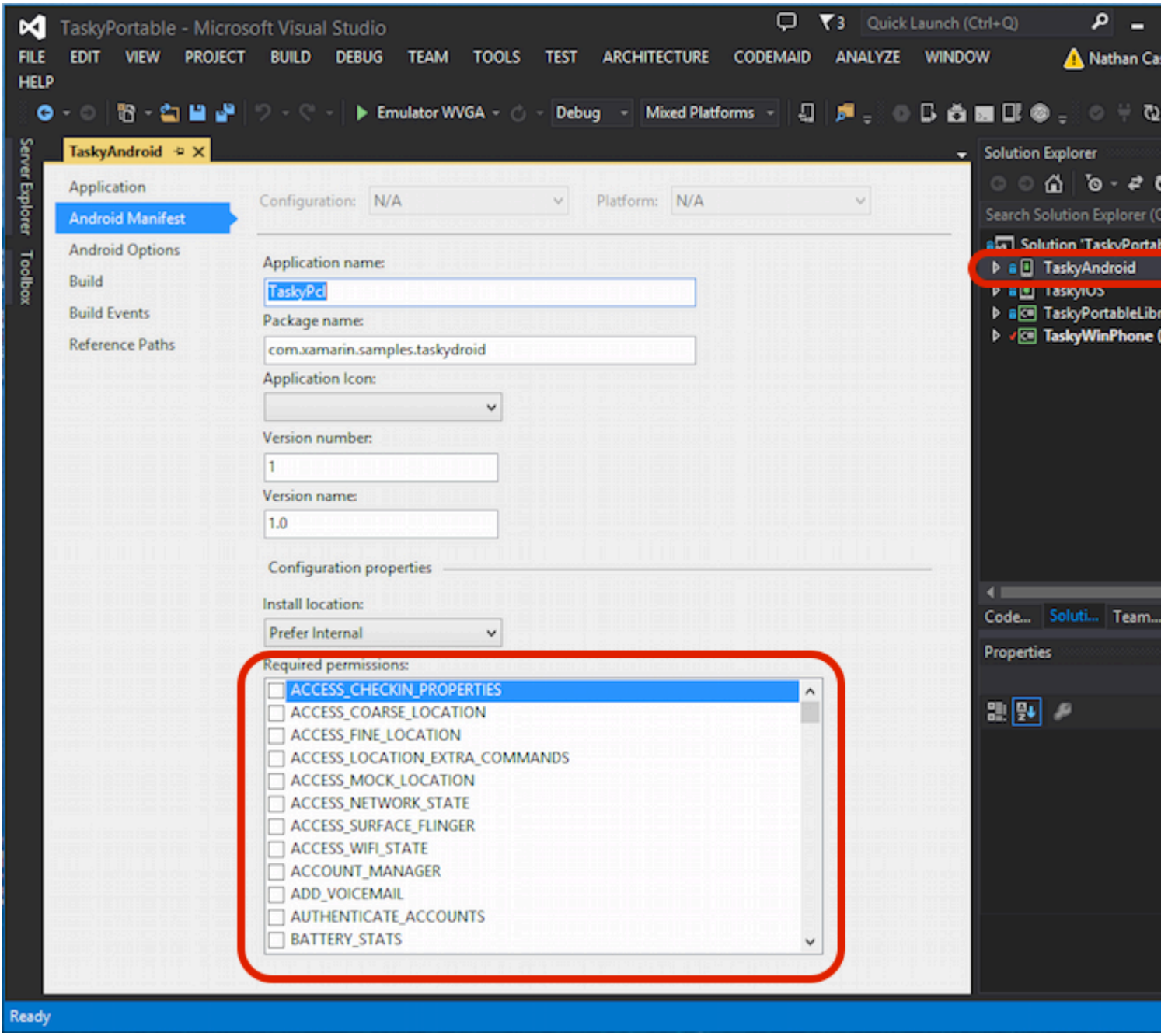
Документация Xamarin обеспечивает хорошую информацию здесь:

https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_applicati

---

**Android Manifest**

Во-первых, в Visual Studio щелкните правой кнопкой мыши проект Xamarin.Android в обозревателе решений и выберите «Свойства». Затем перейдите на вкладку манифеста Android, чтобы увидеть этот экран:




В отличие от Android Studio или Eclipse, вам не нужно установить файл AndroidManifest.xml, написав; Xamarin и Visual Studio делают это за вас. Мероприятия, BroadcastReceivers и Services вставляются в Android Manifest путем объявления определенных атрибутов в своих классах .

На этом экране доступны следующие опции:

- **Имя приложения** : это имя приложения, которое будет отображаться для пользователя.
- **Имя пакета** : это имя пакета. Он должен быть уникальным, что означает, что он не должен использовать одно и то же имя пакета других приложений в Google Play Store.
- **Значок приложения** : это значок, который будет отображаться для пользователя, что эквивалентно @ drawable / ic_launcher, используемому в проектах Android Studio

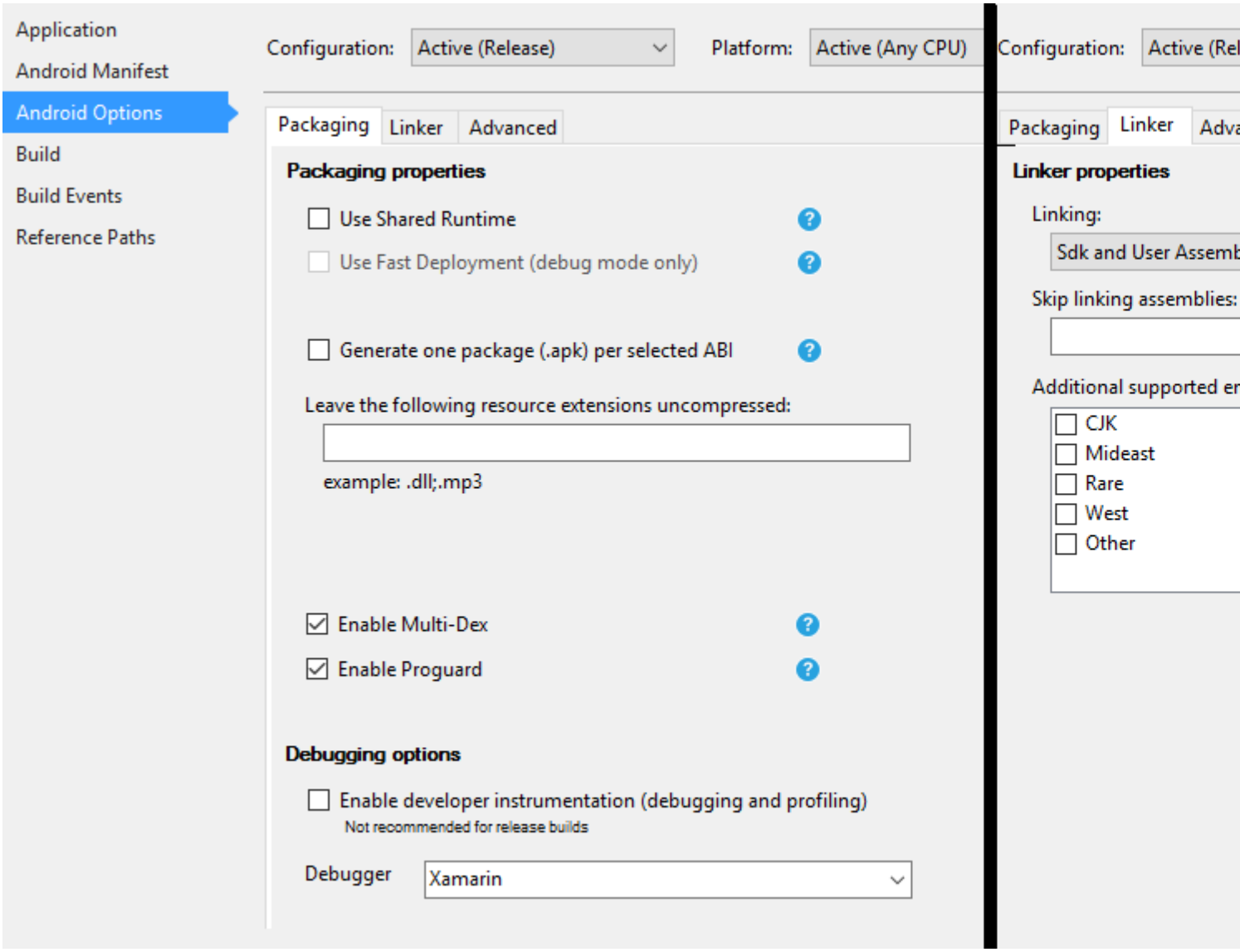
или Eclipse.

- **Номер версии** : **номер** версии используется Google Play для контроля версий. Если вы хотите опубликовать APK для обновленной версии вашего приложения, вы должны добавить 1 к этому номеру для каждого нового обновления.
- **Название версии** : это имя версии, которое будет отображаться для пользователя.
- **Место установки** : определяет, где будет установлен ваш APK, в хранилище устройств или на SD-карте.
- **Необходимые разрешения** : здесь вы определяете, какие разрешения необходимы для вашего приложения.

---

### Настройки Android

На приведенном ниже экране вы можете настроить параметры компилятора. Использование правильных параметров здесь может значительно уменьшить ваш размер APK, а также предотвратить ошибки.




- **Конфигурация** : **Активный (Release)** .

- **Платформа** : **активная (любой процессор)** . Это необходимо для создания APK для магазина Google Play. Если для параметра «Конфигурация» установлено значение «Отладка», он не будет принят в Google Play.
- **Использовать общий Runtime** : **false** . Если вы установите значение true, APK будет использовать Mono Runtime для выполнения. Mono Runtime устанавливается автоматически при отладке через USB, но не в Release APK. Если Mono Runtime не установлено в устройстве, и этот параметр установлен в true в Release APK, приложение выйдет из строя.
- **Создайте один пакет (.apk) для выбранного ABI** : **false** . Создайте APK как можно больше платформ по соображениям совместимости.
- **Включить Multi-Dex** : **true** , но вы можете установить его в false, если ваше приложение не очень сложно (то есть имеет менее 65536 методов, см. Здесь ).
- **Включить Proguard** : **true** . Это позволяет инструменту Proguard обмануть Java-код в вашем приложении. Обратите внимание, что это не относится к .NET-коду; если вы хотите обфускать .NET-код, вы должны использовать Dotfuscator . Более подробную информацию о Proguard для Xamarin.Android можно найти здесь .
- **Включить инструментарий разработчика (отладка и профилирование)** : **false** для Release APK.
- **Связь** : **SDK и пользовательские сборки** . Это заставит Xamarin Linker удалить все неиспользуемые классы из SDK и вашего кода, уменьшив размер APK.

## Важный

Xamarin.Linker иногда может удалять классы, которые, похоже, не используются вашим кодом, особенно если они находятся в ядре проекта (библиотека PCL). Чтобы этого избежать, вы можете установить привязку только к «Sdk Assemblies Only» или использовать атрибут Preserve в своих классах, например:

**PreserveAttribute.cs**

```
namespace My_App_Core.Models
{
    public sealed class PreserveAttribute : System.Attribute
    {
        public bool AllMembers;
        public bool Conditional;
    }
}
```

**В классе:**

```
using System;

namespace My_App_Core.Models
{
    [Preserve(AllMembers = true)]
    public class ServiceException : Exception
```

```
    {
        public int errorCode;

        [Preserve(AllMembers = true)]
        public ServiceException() { }

        [Preserve(AllMembers = true)]
        public ServiceException(int errorCode)
        {
            this.errorCode = errorCode;
        }
    }
}
```

- **Поддерживаемые архитектуры** : **выберите все** по соображениям совместимости.

После настройки всего, перестройте проект, чтобы убедиться, что он успешно работает.

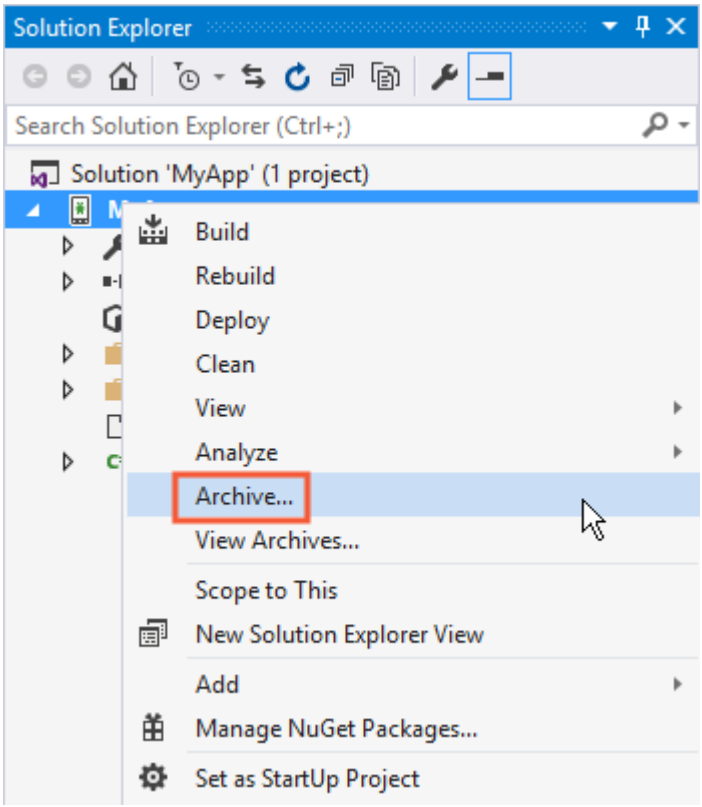
---

### Создание режима APK для выпуска

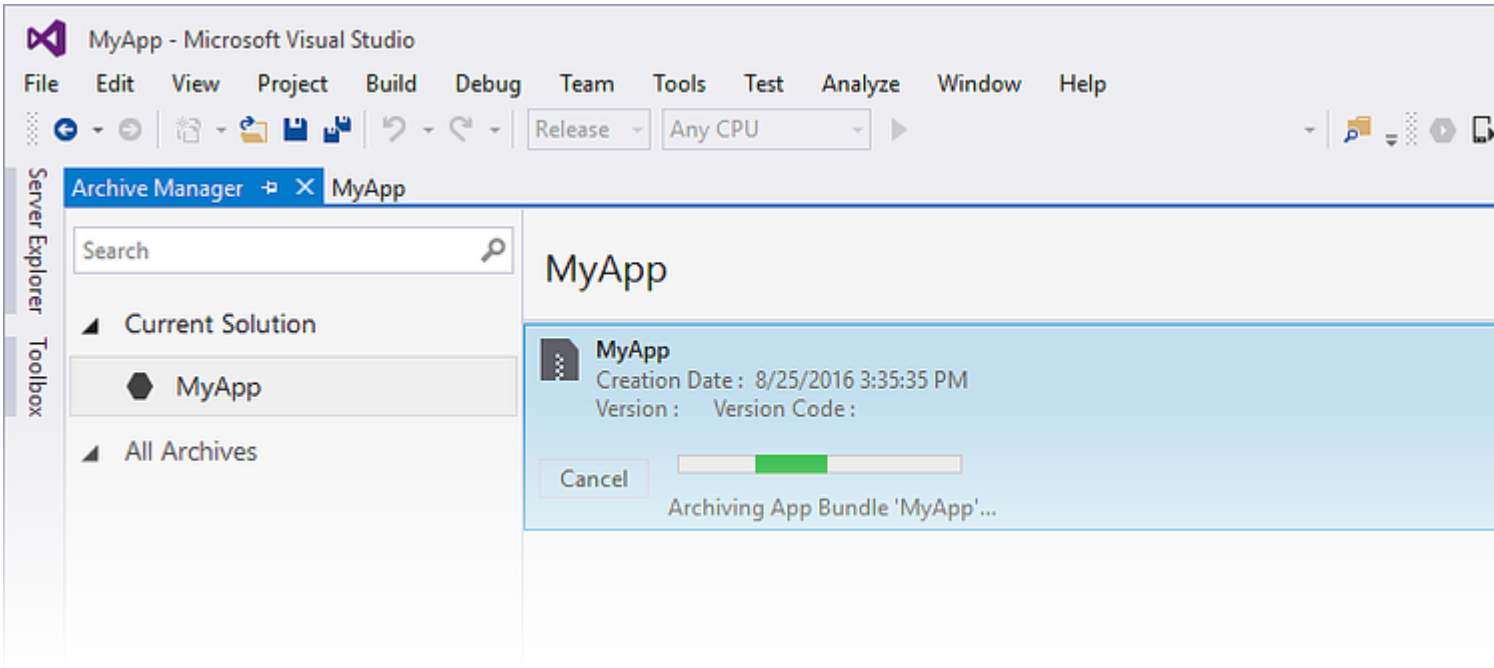
Вы закончили настройку своего Android-проекта для выпуска. В следующем уроке показано, как сгенерировать APK в Visual Studio. Полный учебник по документации Xamarin можно найти здесь:

https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_applicati
_signing_the_android_application_package/

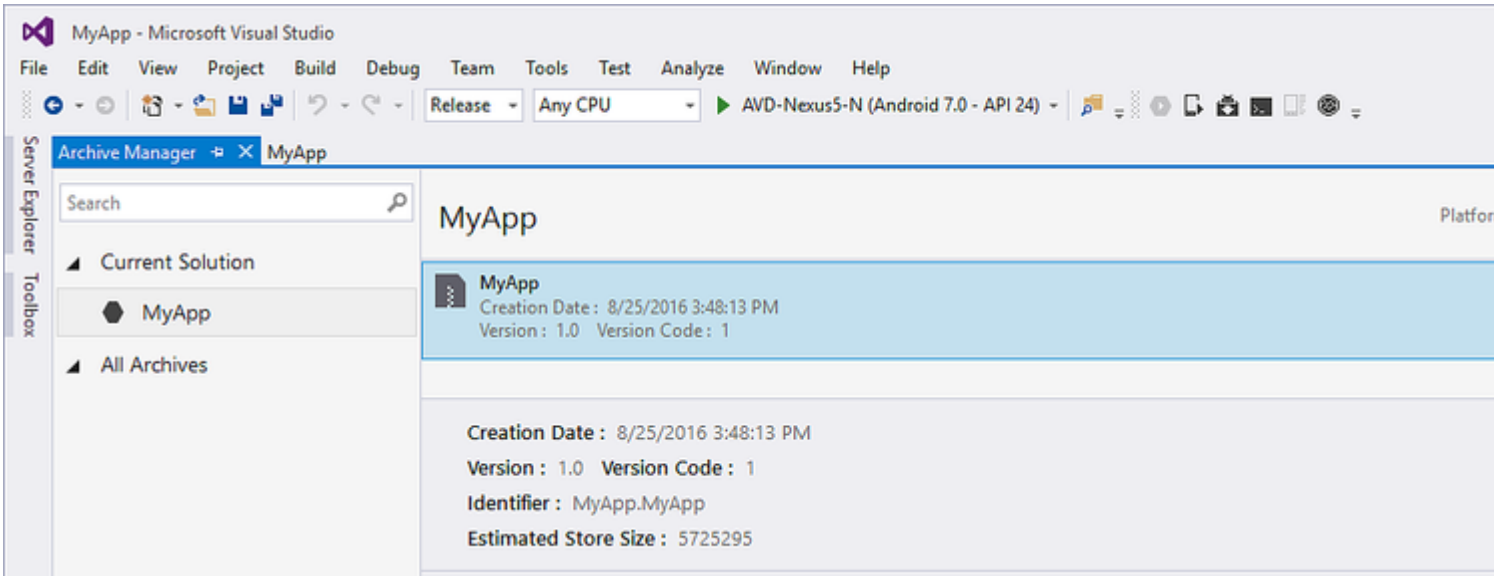
Чтобы создать файл APK, щелкните правой кнопкой мыши проект Xamarin.Android в обозревателе решений и выберите Archive ...



Это откроет диспетчер архива и начнет архивирование проекта, готовясь к созданию файла APK.

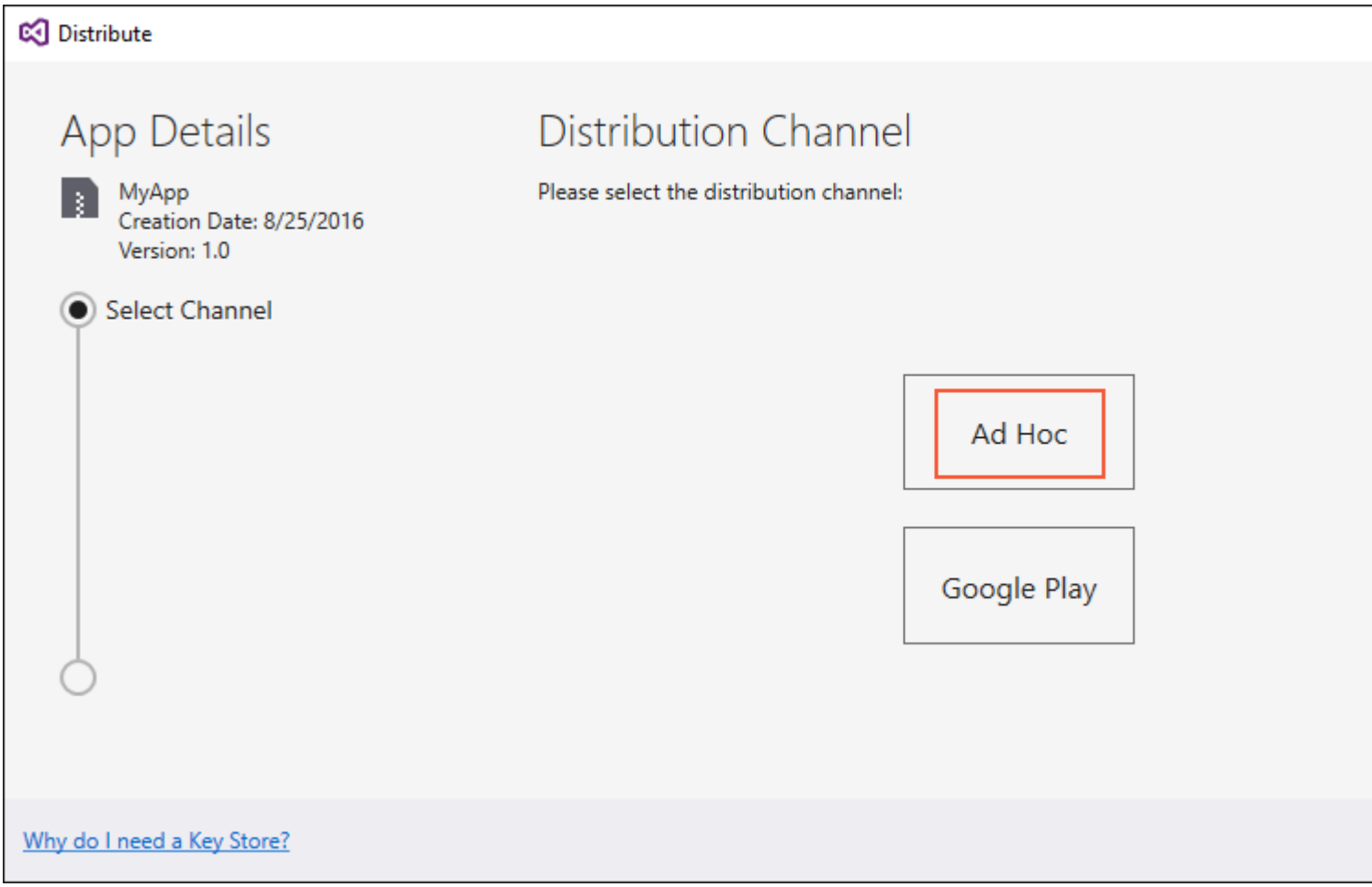



Когда он завершит архивирование проекта, нажмите «Распространять» ... чтобы продолжить.

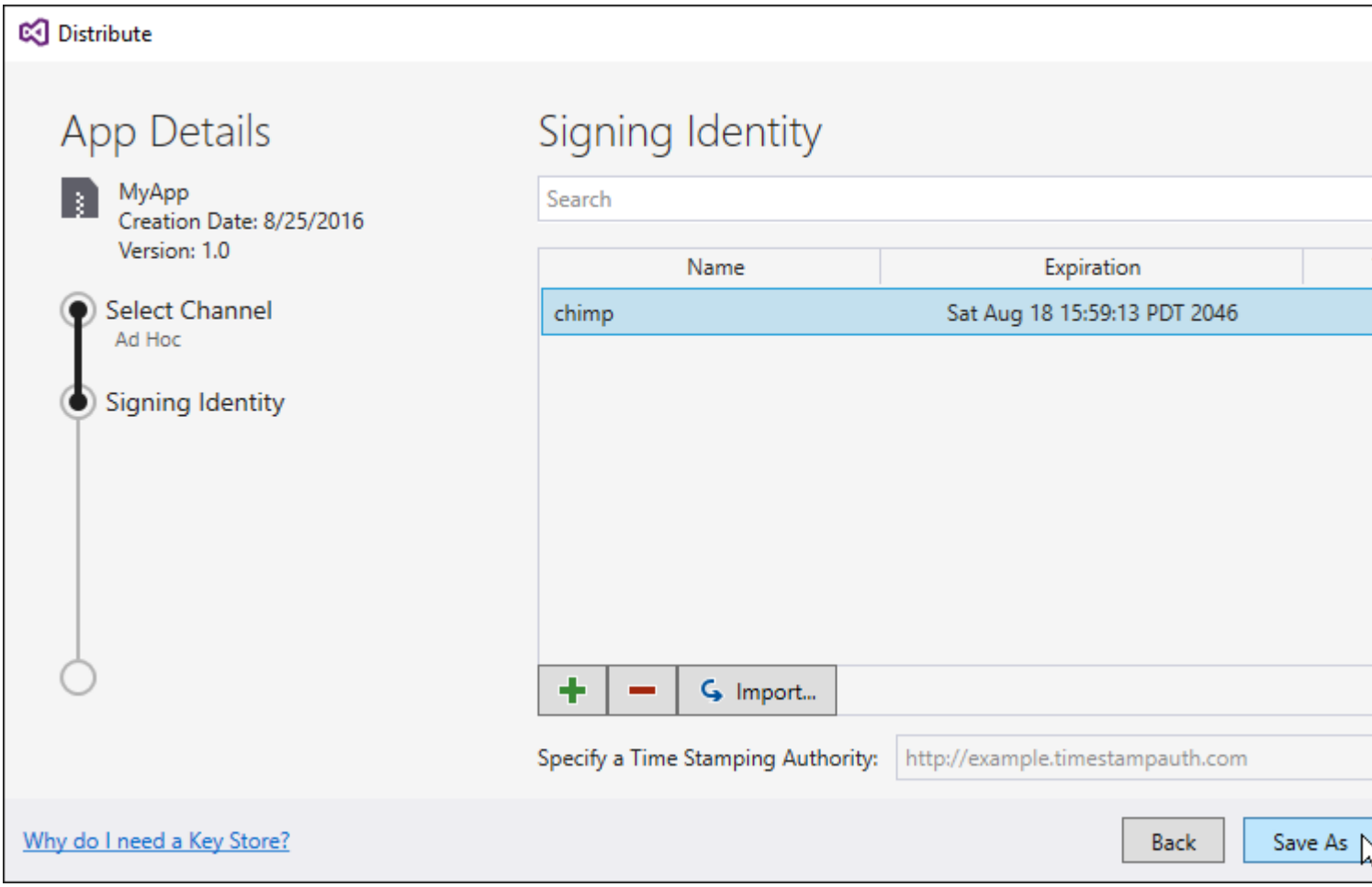



На экране «Распределить» будут представлены два варианта: «Ad-hoc» и «Google Play». Первый создаст APK и сохранит его на вашем компьютере. Второй будет публиковать приложение прямо в Google Play.

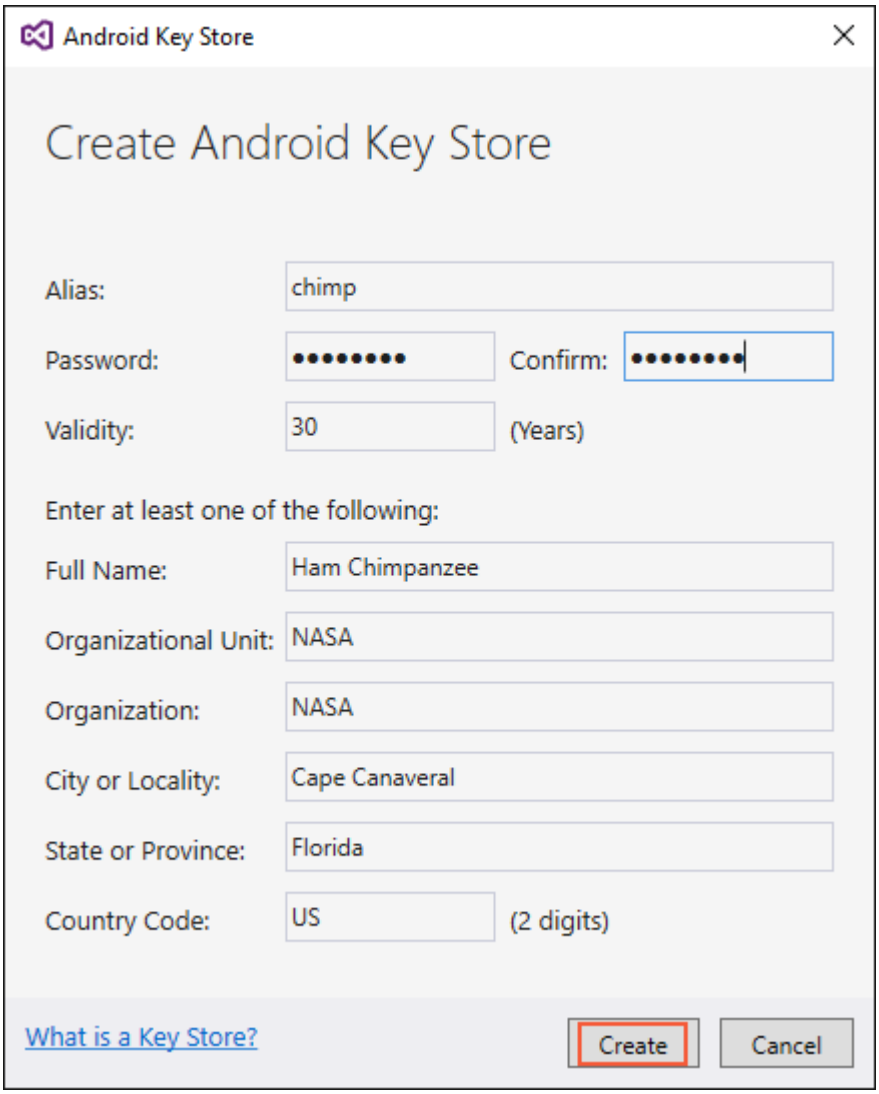
Рекомендуется выбрать первый, поэтому вы можете протестировать APK на других устройствах, если хотите.



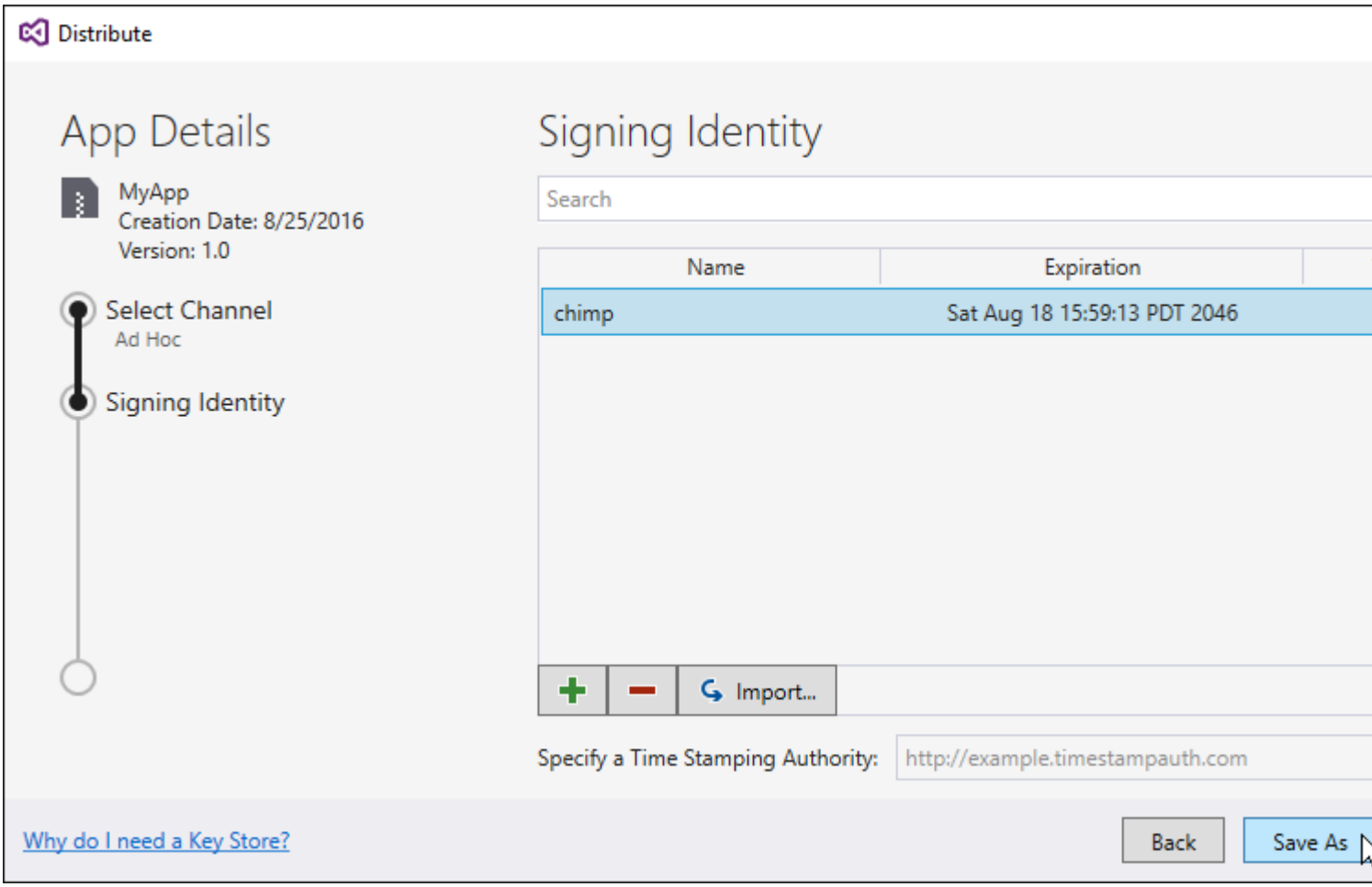

На следующем экране для подписывания APK необходим магазин Android Key. Если у вас уже есть, вы можете использовать его, нажав «Импортировать ...»; если вы этого не сделаете, вы можете создать новый Android Key Store, нажав +.

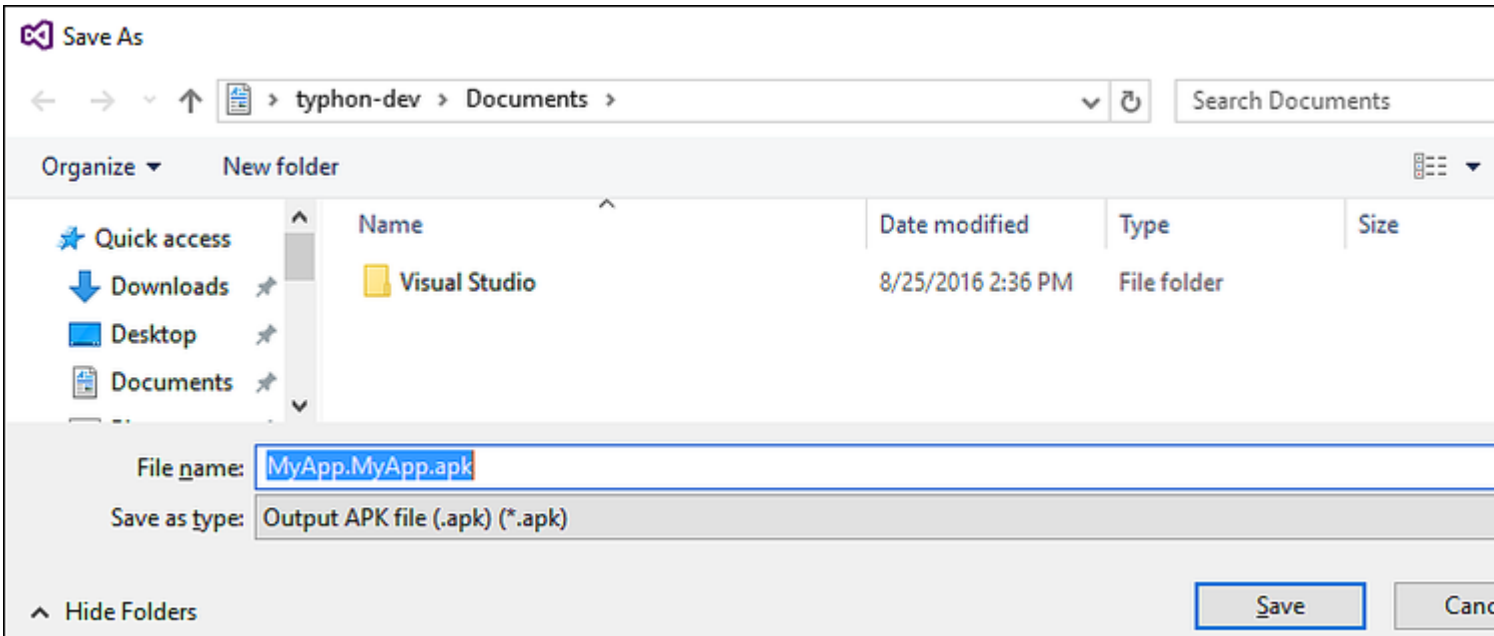


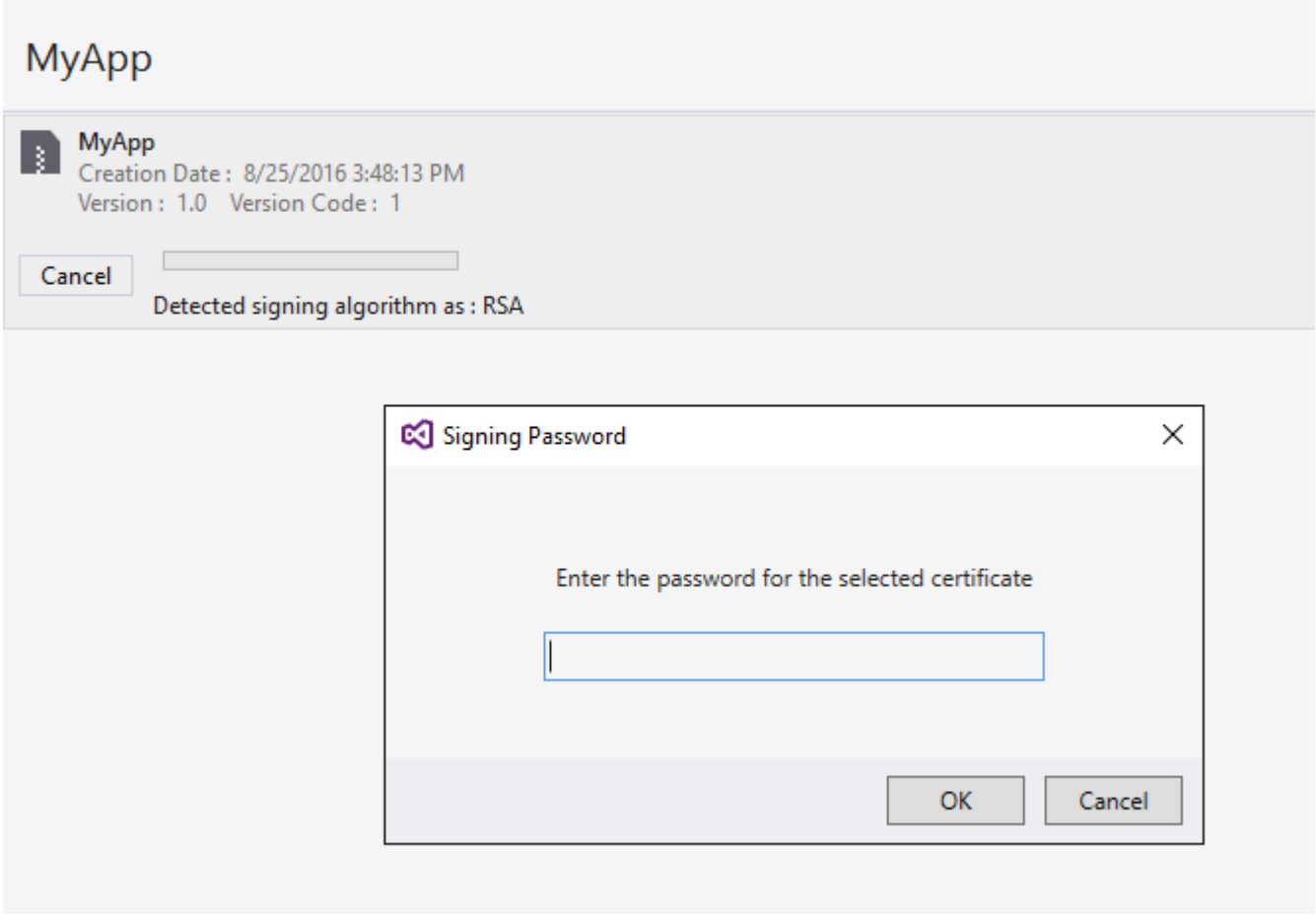

Создание нового экрана Android Key Store:



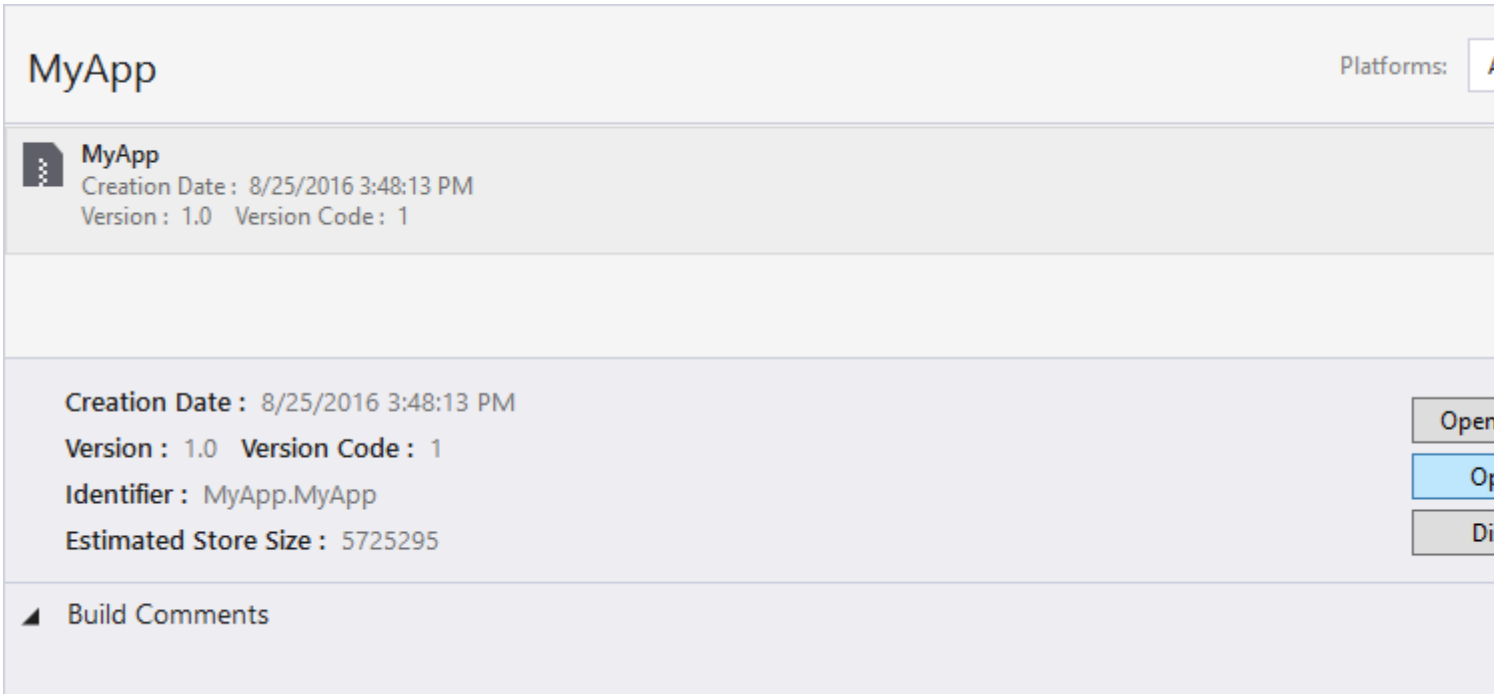

Чтобы создать APK, нажмите «Сохранить как». Возможно, вам будет предложено ввести пароль хранилища ключей.

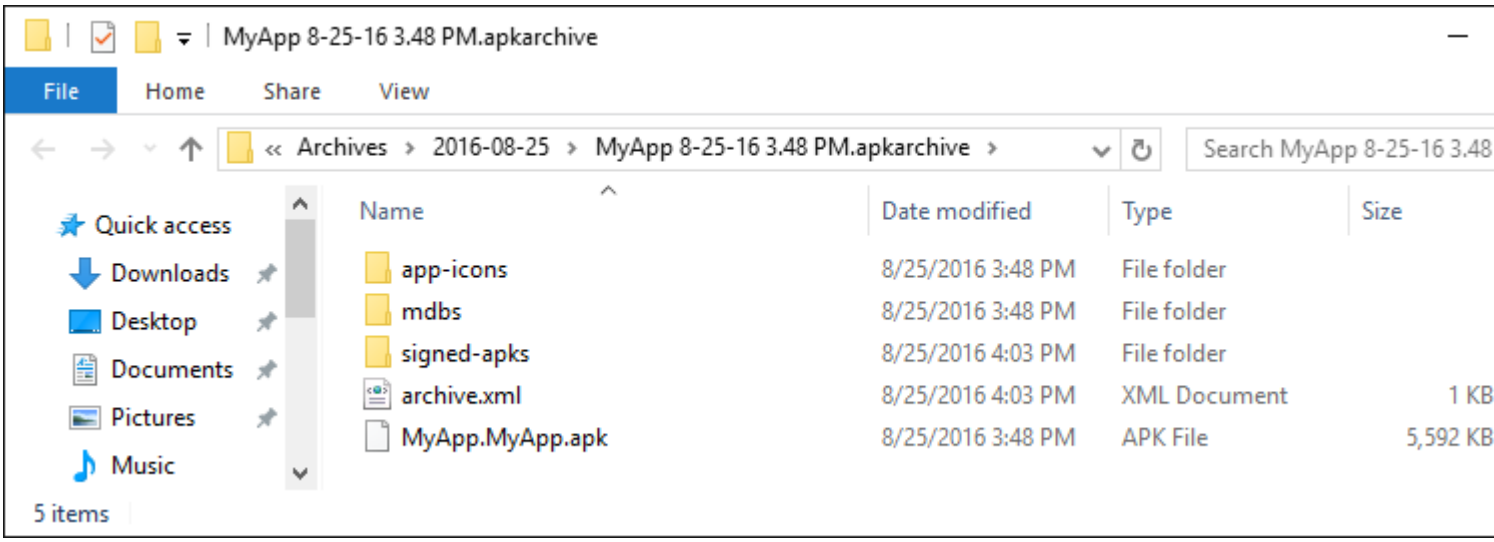

Distribute
App Details
MyApp
Creation Date: 8/25/2016
Version: 1.0
Select Channel
Ad Hoc
Signing Identity
Signing Identity
Search
Name
Expiration
chimp
Sat Aug 18 15:59:13 PDT 2046
Import...
Specify a Time Stamping Authority:
http://example.timestampauth.com
Why do I need a Key Store?
Back
Save As


Save As
typhon-dev
Documents
Search Documents
Organize
New folder
Quick access
Downloads
Desktop
Documents
Name
Date modified
Type
Size
Visual Studio
8/25/2016 2:36 PM
File folder
File name:
MyApp.MyApp.apk
Save as type:
Output APK file (.apk) (*.apk)
Hide Folders
Save




По завершении нажмите «Открыть папку» на экране «Архивы», чтобы увидеть сгенерированный файл APK.






### Включение MultiDex в вашем Xamarin.Android APK

MultiDex - это библиотека в Android APK, которая позволяет приложению иметь более 65 536 методов.

В Android APK есть исполняемые файлы Dalvik (.dex), которые содержат сгенерированные байт-коды, скомпилированные из вашего Java-кода. Каждый файл .dex может содержать до 65 536 методов (2 ^ 16).

В версиях ОС Android до Android 5.0 Lollipop (API 21) используется среда исполнения Dalvik, которая поддерживает только один файл .dex на APK, ограничивая 65 536 методов на APK. Начиная с Android 5.0, ОС Android использует время выполнения ART, которое может поддерживать более одного файла .dex на APK, избегая ограничений.

Чтобы превзойти предел 65k-методов в версиях Android ниже API 21, разработчики должны использовать библиотеку поддержки MultiDex. MultiDex создает дополнительные файлы classes.dex (classes2.dex, classes3.dex, ...), ссылаясь на них в файле classes.dex. Когда приложение начинает загрузку, он использует класс MultiDexApplication для загрузки дополнительных файлов .dex.

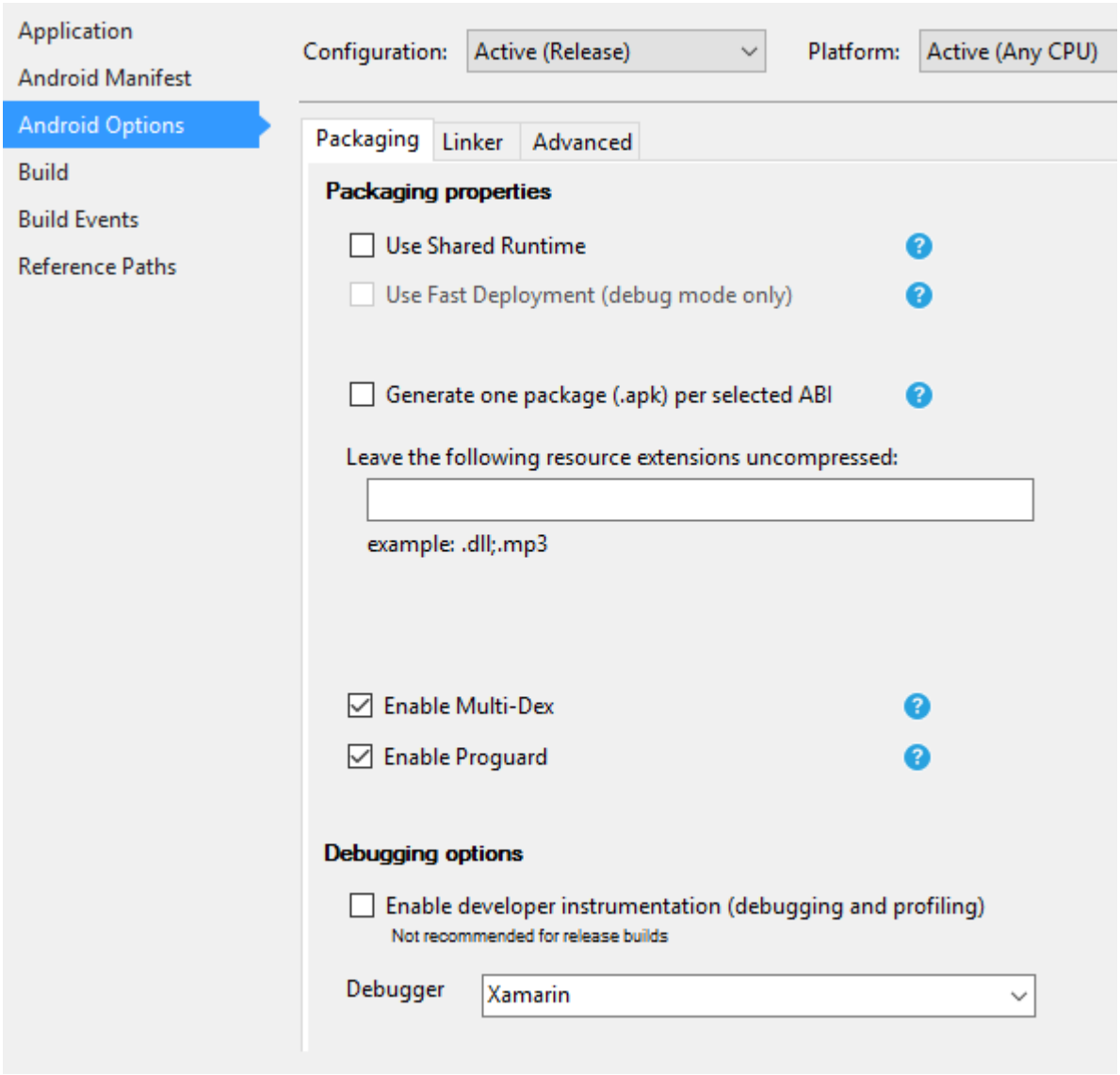
Если ваше приложение для Android нацелено на минимальную версию SDK выше или равную API 21 (Android 5.0 Lollipop), нет необходимости использовать библиотеку MultiDex, потому что ОС обрабатывает изначально дополнительные файлы .dex. Однако, если по соображениям совместимости разработчик хочет поддерживать устаревшую ОС Android, он должен использовать библиотеку MultiDex.

---

## Как использовать MultiDex в приложении Xamarin.Android

Во-первых, чтобы включить MultiDex в приложении Xamarin.Android, перейдите в свой

проект «Свойства» -> «Настройки Android» -> «Упаковка» -> «Включить многоэкранный режим», как на экране печати ниже:




Затем вы должны создать класс MultiDexApplication в своем приложении. В корне проекта создайте новый класс (в обозревателе решений щелкните правой кнопкой мыши в проекте Add .. -> Class или Shift + Alt + C). В новом файле класса скопируйте следующий код, заменив пространство имен Sample на имя пространства имен проектов Xamarin.Android.

```
using System;
using Android.App;
using Android.Runtime;
using Java.Interop;

namespace Sample
{
    [Register("android/support/multidex/MultiDexApplication", DoNotGenerateAcw = true)]
    public class MultiDexApplication : Application
    {
        internal static readonly JniPeerMembers _members =
        new XAPeerMembers("android/support/multidex/MultiDexApplication", typeof
(MultiDexApplication));
```

```
        internal static IntPtr java_class_handle;

        private static IntPtr id_ctor;

        [Register(".ctor", "()V", "", DoNotGenerateAcw = true)]
        public MultiDexApplication()
        : base(IntPtr.Zero, JniHandleOwnership.DoNotTransfer)
        {
            if (Handle != IntPtr.Zero)
                return;

            try
            {
                if (GetType() != typeof (MultiDexApplication))
                {
                    SetHandle(
                        JNIEnv.StartCreateInstance(GetType(), "()V"),
                        JniHandleOwnership.TransferLocalRef);
                        JNIEnv.FinishCreateInstance(Handle, "()V");
                    return;
                }

                if (id_ctor == IntPtr.Zero)
                    id_ctor = JNIEnv.GetMethodID(class_ref, "<init>", "()V");
                SetHandle(
                    JNIEnv.StartCreateInstance(class_ref, id_ctor),
                    JniHandleOwnership.TransferLocalRef);
                JNIEnv.FinishCreateInstance(Handle, class_ref, id_ctor);
            }
            finally
            {
            }
        }

        protected MultiDexApplication(IntPtr javaReference, JniHandleOwnership transfer)
            : base(javaReference, transfer)
        {
        }

        internal static IntPtr class_ref
        {
            get { return JNIEnv.FindClass("android/support/multidex/MultiDexApplication", ref
java_class_handle); }
        }

        protected override IntPtr ThresholdClass
        {
            get { return class_ref; }
        }

        protected override Type ThresholdType
        {
            get { return typeof (MultiDexApplication); }
        }
    }
}
```

Источник кода здесь.

Если вы разрабатываете Visual Studio для Windows, также есть ошибка в инструментах

сборки Android SDK, которые необходимо исправить, чтобы правильно создавать файлы classes.dex при создании проекта.

Перейдите в папку Android SDK, откройте папку с инструментами создания и установите папки с номерами компиляторов Android SDK, например:

C: \ Android-SDK \ сборки-инструменты \ 23.0.3 \

C: \ Android-SDK \ сборки-инструменты \ 24.0.1 \

C: \ Android-SDK \ сборки-инструменты \ 25.0.2 \

Внутри каждой из этих папок есть файл **mainClassesDex.bat** , пакетный скрипт, используемый для создания файлов classes.dex. Откройте каждый файл mainClassesDex.bat с помощью текстового редактора (Notepad или Notepad ++) и в его скрипте найдите и замените блок:

```
if DEFINED output goto redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
"%disableKeepAnnotated%" "%tmpJar%" "%params%"
goto afterClassReferenceListBuilder
:redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
"%disableKeepAnnotated%" "%tmpJar%" "%params%" 1>"%output%"
:afterClassReferenceListBuilder
```

С блоком:

```
SET params=%params:'=%
if DEFINED output goto redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
%disableKeepAnnotated% "%tmpJar%" %params%
goto afterClassReferenceListBuilder
:redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
%disableKeepAnnotated% "%tmpJar%" %params% 1>"%output%"
:afterClassReferenceListBuilder
```

Источник здесь.

Сохраните каждый mainClassesDex.bat в текстовом редакторе после изменений.

После описанных выше шагов вы сможете успешно создать приложение Xamarin.Android с помощью MultiDex.

## Включение ProGuard в ваш Xamarin.Android APK

ProGuard - это инструмент, используемый в процессе построения для оптимизации и обфускации кода Java вашего APK, а также удаления неиспользуемых классов. Полученный APK при использовании ProGuard будет иметь меньший размер и будет

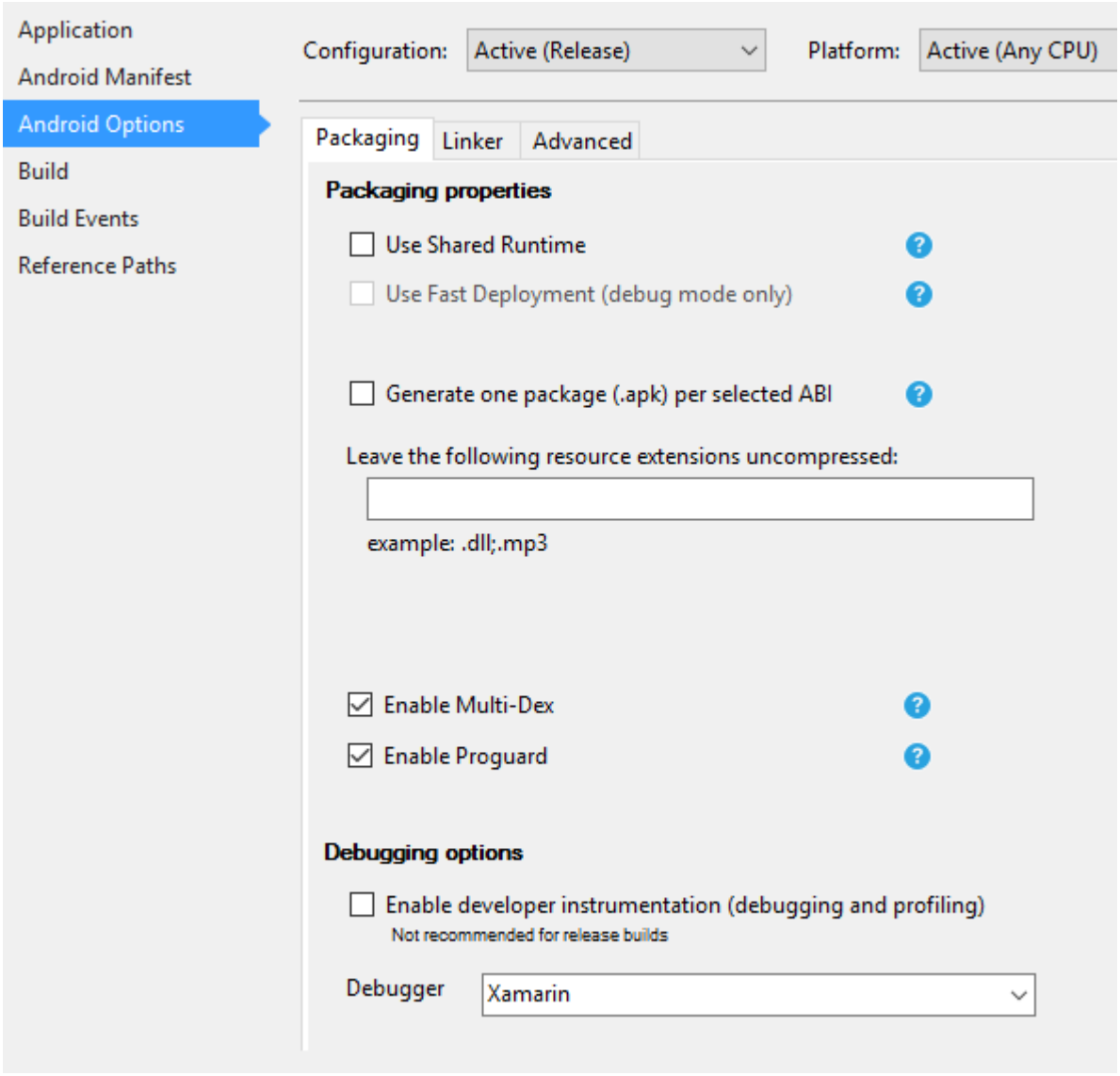
сложнее перепроектировать (декомпиляция).

ProGuard можно использовать также в приложениях Xamarin.Android, а также уменьшит размер файла APK и запутывает Java-код. Однако имейте в виду, что обфускация ProGuard применяется только к Java-коду. Чтобы запутать .NET-код, разработчику следует использовать Dotfuscator или аналогичные инструменты.

---

## Как использовать ProGuard в приложении Xamarin.Android

Во-первых, чтобы включить ProGuard в вашем приложении Xamarin.Android, перейдите в свой проект Properties -> Android Options -> Packaging -> Включите ProGuard, как на экране печати ниже:

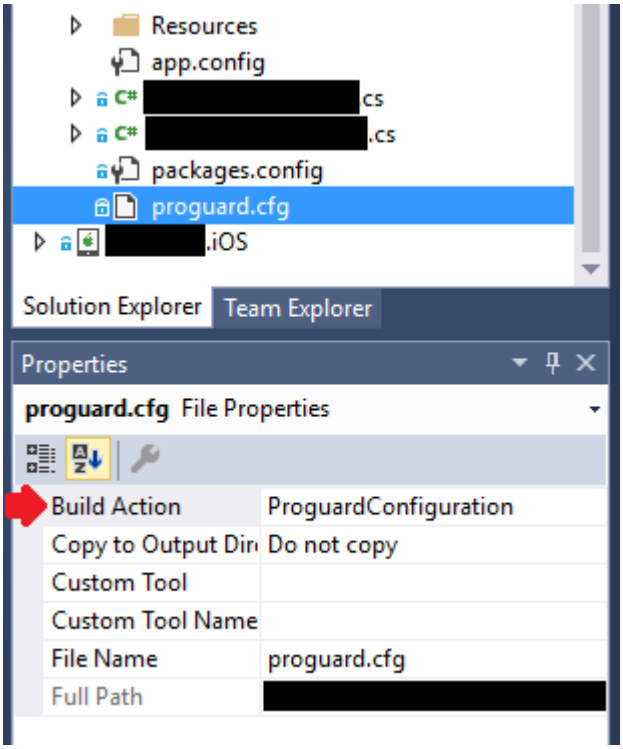



Это позволяет ProGuard при создании приложения.

Xamarin.Android по умолчанию устанавливает свои собственные конфигурации для

ProGuard, которые можно найти внутри папок `obj/Debug/proguard` или `obj/Release/proguard` в файлах `proguard_project_primary.cfg` , `proguard_project_references.cfg` и `proguard_xamarin.cfg` . Эти три файла объединены в качестве конфигураций для ProGuard, и они автоматически создаются Xamarin при построении.

Если разработчик хочет дополнительно настроить параметры ProGuard, он может создать файл в корне проекта с именем `proguard.cfg` (другие имена также действительны, если расширение является .cfg) и устанавливают его действие Build на ProguardConfiguration, как на рисунке ниже:




В файле могут быть вставлены пользовательские параметры ProGuard, такие как `-dontwarn` , `-keep class` и другие .

**Важный**

Как и сейчас (апрель / 2017), Android SDK, который обычно загружается, имеет старую версию ProGuard, которая может вызвать ошибки при создании приложения с использованием Java 1.8. При построении в списке ошибок отображается следующее сообщение:

```
Error
Can't read [C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\MonoAndroid\v7.0\mono.android.jar]
(Can't process class [android/app/ActivityTracker.class]  (Unsupported class version number
[52.0] (maximum 51.0, Java 1.7))) [CREATEMULTIDEXMAINDEXCLASSLIST]
```

Источник здесь.

Чтобы устранить эту проблему, вы должны загрузить последнюю версию ProGuard ( здесь )

и скопировать содержимое .zip-файла в `android-sdk\tools\proguard\` . Это позволит обновить ProGuard, и процесс сборки должен работать без проблем.

После этого вы сможете успешно создать приложение Xamarin.Android с помощью ProGuard.

### «Таинственные» ошибки, связанные с ProGuard и Linker

Вы сделали отличное приложение и протестировали его в Debug с хорошими результатами. Все работает нормально!

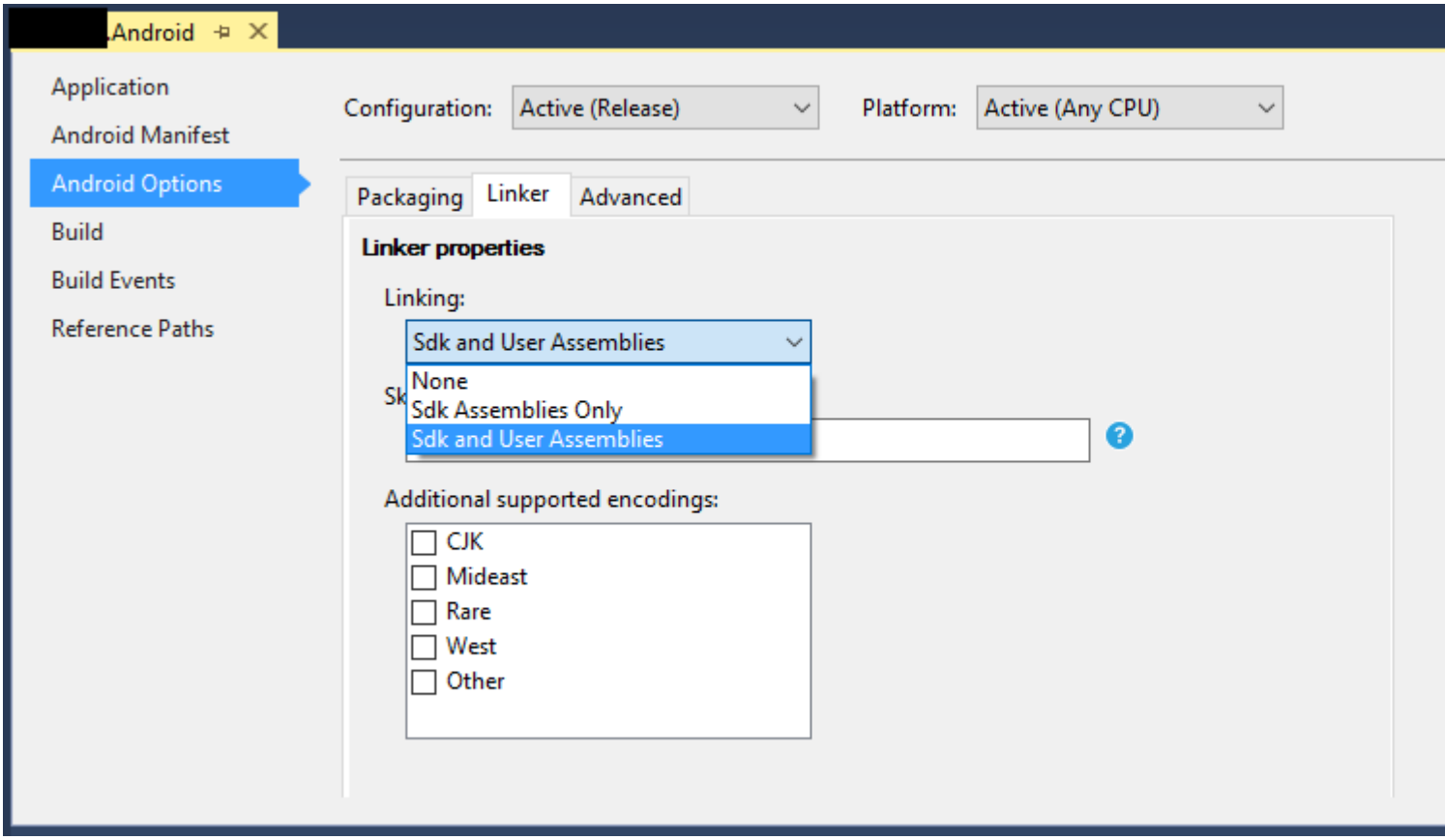
Но тогда вы решили подготовить свое приложение для выпуска. Вы создали MultiDex, ProGuard и Linker, а затем перестали работать.

Этот учебник призван помочь вам найти общие проблемы, связанные с ProGuard и Linker, которые могут вызвать загадочные ошибки.

---

## Понимание Xamarin.Linker

Xamarin.Linker - это инструмент в процессе построения, который удаляет неиспользуемый код и классы **из вашего кода .NET (а не кода Java)** . В свойствах вашего проекта -> Настройки Android -> Линкера появится окно выбора. Ссылка на параметры:




**Нет** : код не удаляется.

**Sdk Assemblies Only** : этот параметр позволяет Xamarin.Linker проверять неиспользуемый код только в библиотеках Xamarin. **Этот параметр безопасен.**

**Sdk и User Assemblies** : этот параметр позволяет Xamarin.Linker проверять неиспользуемый код в библиотеках Xamarin и в коде проекта (включая PCL, компоненты Xamarin и пакеты NuGet). **Этот вариант не всегда безопасен!**

При использовании опций Sdk и User Assemblies Xamarin.Linker может думать, что части кода не используются, когда на самом деле они очень используются! Это может привести к тому, что некоторые библиотеки перестанут работать должным образом и вызывают ошибки в вашем приложении.

Чтобы Xamarin.Linker не удалял код, есть 3 варианта:

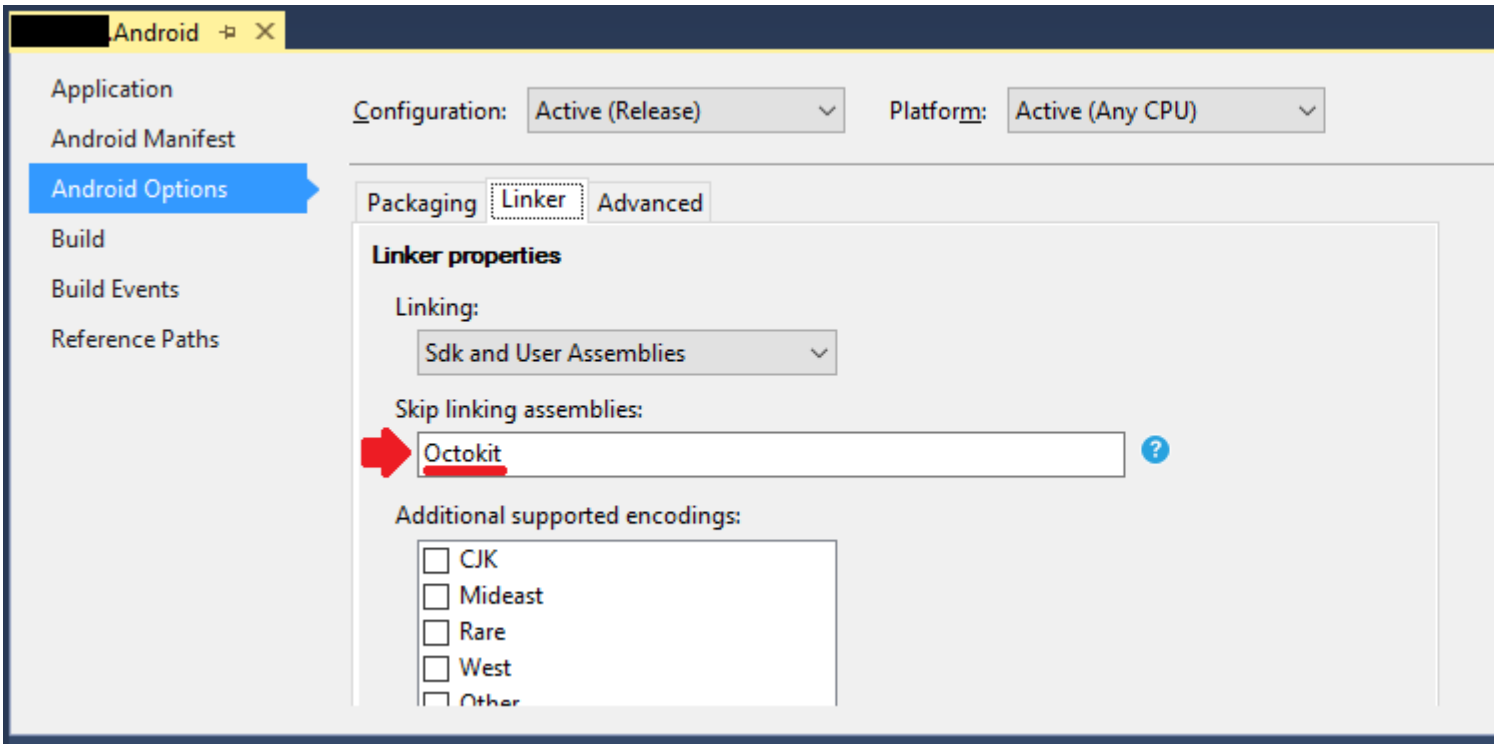
1. Установка параметра «Связывание» на «Нет» или «Только Sdk»;
2. Пропустить соединительные сборки;
3. Использование атрибута Preserve.

**Пример для 2. Пропустить соединительные сборки:**

В приведенном ниже примере использование Xamarin.Linker вызвало пакет NuGet ( Octokit ), который отлично работает, чтобы перестать работать, потому что он больше не может подключаться к Интернету:

```
[0:] ERROR
[0:] SOURCE: mscorlib
[0:] MESSAGE: Object reference not set to an instance of an object.
[0:] STACK TRACE:   at Octokit.PocoJsonSerializerStrategy.DeserializeObject (System.Object
value, System.Type type) [0x003d8] in D:\repos\octokit.net\Octokit\SimpleJson.cs:1472
  at Octokit.Internal.SimpleJsonSerializer+GitHubSerializerStrategy.DeserializeObject
(System.Object value, System.Type type) [0x001c3] in
D:\repos\octokit.net\Octokit\Http\SimpleJsonSerializer.cs:165
  at Octokit.SimpleJson.DeserializeObject (System.String json, System.Type type,
Octokit.IJsonSerializerStrategy jsonSerializerStrategy) [0x00007] in
D:\repos\octokit.net\Octokit\SimpleJson.cs:583
  at Octokit.SimpleJson.DeserializeObject[T] (System.String json,
Octokit.IJsonSerializerStrategy jsonSerializerStrategy) [0x00000] in
D:\repos\octokit.net\Octokit\SimpleJson.cs:595
  at Octokit.Internal.SimpleJsonSerializer.Deserialize[T] (System.String json) [0x00000] in
D:\repos\octokit.net\Octokit\Http\SimpleJsonSerializer.cs:21
  at Octokit.Internal.JsonHttpPipeline.DeserializeResponse[T] (Octokit.IResponse response)
[0x000a7] in D:\repos\octokit.net\Octokit\Http\JsonHttpPipeline.cs:62
  at Octokit.Connection+<Run>d__54`1[T].MoveNext () [0x0009c] in
D:\repos\octokit.net\Octokit\Http\Connection.cs:574
--- End of stack trace from previous location where exception was thrown ---
```

Чтобы библиотека снова начала работать, необходимо было добавить имя ссылки пакета в поле «Связывание сборок», расположенное в проекте -> «Свойства» -> «Настройки Android» -> «Линкер», как показано на рисунке ниже:




После этого библиотека начала работать без каких-либо проблем.

**Пример для 3. Использование атрибута Preserve:**

Xamarin.Linker воспринимает как неиспользуемый код, в основном, код классов моделей в ядре вашего проекта.

Чтобы класс сохранялся в процессе связывания, вы можете использовать атрибут Preserve.

Во-первых, создайте в своем ядре проекта класс **PreserveAttribute.cs** , вставьте следующий код и замените пространство имен пространством имен вашего проекта:

PreserveAttribute.cs:

```
namespace My_App_Core.Models
{
    public sealed class PreserveAttribute : System.Attribute
    {
        public bool AllMembers;
        public bool Conditional;
    }
}
```

В каждом модельном классе ядра вашего проекта вставьте атрибут Preserve, как в приведенном ниже примере:

Country.cs:

```
using System;
using System.Collections.Generic;
```

```
namespace My_App_Core.Models
{
    [Preserve(AllMembers = true)]
    public class Country
    {
        public String name { get; set; }
        public String ISOcode { get; set; }

        [Preserve(AllMembers = true)]
        public Country(String name, String ISOCode)
        {
            this.name = name;
            this.ISOCode = ISOCode;
        }
    }
}
```

После этого процесс связывания больше не удалит сохраненный код.

---

# Понимание ProGuard

ProGuard - это инструмент в процессе построения, который удаляет неиспользуемый код и классы **из вашего Java-кода** . Он также запутывает и оптимизирует код.

Однако ProGuard иногда может удалить код, который он воспринимает как неиспользуемый, когда это не так. Чтобы этого избежать, разработчик должен отладить приложение (в Android Device Monitor и Visual Studio Debug) и определить, какой класс был удален, а затем настроить конфигурационный файл ProGuard для сохранения класса.

**пример**

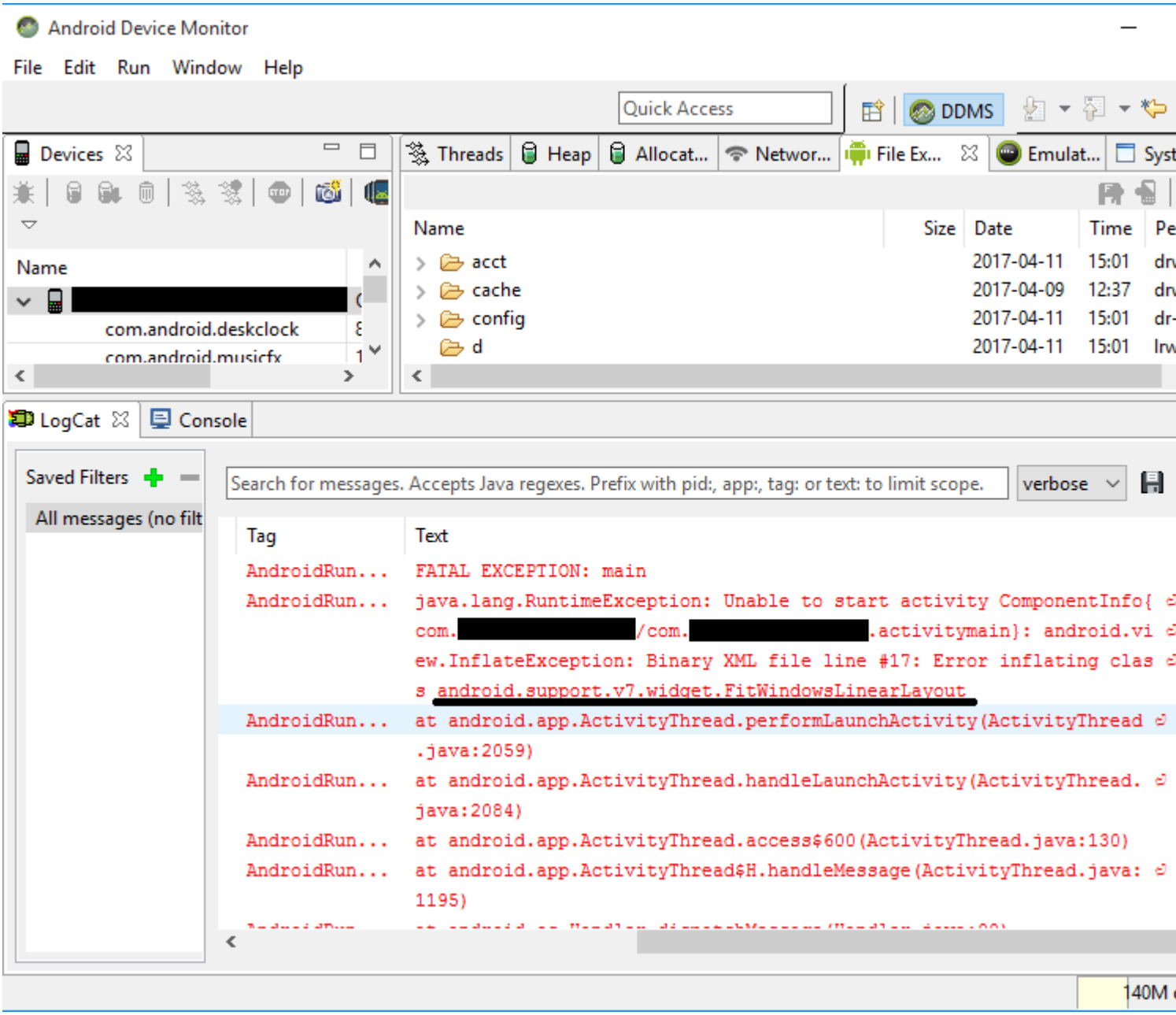
В приведенном ниже примере ProGuard удалил два класса ( Android.Support.V7.Widget.FitWindowsLinearLayout и Android.Support.Design.Widget.AppBarLayout), используемые в файлах макетов AXML, но которые были восприняты как неиспользуемые в коде. При удалении вызвало ClassNotFoundException в коде Java при отображении макета активности:

layout_activitymain.axml:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activitymain_drawerlayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true" <!-- ### HERE ### -->
    tools:openDrawer="start">
    <RelativeLayout
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true">
        <!-- ### HERE ## -->
        <android.support.design.widget.AppBarLayout
            android:id="@+id/activitymain_appbarlayout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/AppTheme.AppBarOverlay">
...
```

LogCat показывает ошибку при создании макета в SetContentView:




Чтобы исправить эту ошибку, необходимо было добавить следующие строки в файл конфигурации ProGuard проекта:

```
-keep public class android.support.v7.widget.FitWindowsLinearLayout
-keep public class android.support.design.widget.AppBarLayout
```

После этого при создании макета больше не было ошибок.

**Предупреждения ProGuard**

ProGuard иногда показывает предупреждения в списке ошибок после создания вашего проекта. Хотя они поднимают вопрос о том, хорошо ли ваше приложение или нет, не все их предупреждения указывают на проблемы, особенно если ваше приложение успешно создает.

Одним из примеров этого является использование библиотеки Picasso : при использовании ProGuard это может показывать предупреждения, такие как `okio.Okio: can't find referenced class (...)` или `can't write resource [META-INF/MANIFEST.MF] (Duplicate zip entry [okhttp.jar:META-INF/MANIFEST.MF]) (...)` , но приложение работает, и библиотека работает без проблем.

Прочитайте Публикация вашего Xamarin.Android APK онлайн:
https://riptutorial.com/ru/xamarin-android/topic/9601/публикация-вашего-xamarin-android-apk

# глава 12: Сканирование штрих-кода с использованием библиотеки ZXing в приложениях Xamarin

## Вступление

Библиотека Zxing хорошо известна для обработки изображений. Zxing был основан на Java, а также доступен модуль .Net, который можно использовать в приложениях xamarin. нажмите здесь, чтобы проверить официальную документацию. http://zxingnet.codeplex.com/

Недавно я использовал этот libarry.

Шаг 1: добавьте компонент ZXing.Net.Mobile в решение.

step2: В зависимости от того, какую активность нам нужно показать сканер штрих-кода, в этой активности инициализируйте MobileBarcodeScanner.

Шаг 3: напишите ниже код, если он используется в любом представлении, чтобы начать сканирование.

## Examples

### Образец кода

```
button.Click +=async delegate
{
var MScanner = new MobileBarcodeScanner();
var Result = await MScanner.Scan();
if(Result == null)
{
    return;
}
//get the bar code text here
string BarcodeText = Result.text;
}
```

Прочитайте Сканирование штрих-кода с использованием библиотеки ZXing в приложениях Xamarin онлайн: https://riptutorial.com/ru/xamarin-android/topic/9526/сканирование-штрих-кода-с-использованием-библиотеки-zxing-в-приложениях-xamarin

# глава 13: Тосты

## Examples

### Основные сообщения для тостов

Сначала `MakeText()` объект Toast с помощью одного из методов `MakeText()` . Этот метод принимает три параметра: приложение `Context` , текстовое сообщение и продолжительность для тоста. Он возвращает правильно инициализированный объект Toast. Вы можете отобразить уведомление тоста с помощью `Show()` , как показано в следующем примере:

```
Context context = Application.Context;
string text = "Hello toast!";
ToastLength duration = ToastLength.Short;

var toast = Toast.MakeText(context, text, duration);
toast.Show();
```

Этот пример демонстрирует все, что вам нужно для большинства уведомлений о тостах. Вам редко нужно что-то еще. Однако вы можете захотеть поместить тост по-другому или даже использовать свой собственный макет вместо простого текстового сообщения. В следующих разделах описывается, как вы можете это делать.

Вы также можете объединить свои методы, вызвать как однострочный и избежать удержания объекта Toast, например:

```
Toast.MakeText(Application.Context, "Hello toast!", ToastLength.Short).Show();
```

Для получения дополнительной информации обратитесь к более полной документации для Android по этой теме.

### Цветные сообщения с тостами

Иногда мы хотим предоставить дополнительную информацию нашему пользователю с цветами (например, красный означает, что что-то случилось неправильно). Мы можем изменить цвет фона тоста, установив цветной фильтр на представление, которое нам дает наш тост (здесь я использую ColorMatrixColorFilter ):

```
Toast t = Toast.MakeText(context, message, duration);
Color c = */your color/*;
ColorMatrixColorFilter CM = new ColorMatrixColorFilter(new float[]
    {
        0,0,0,0,c.R,
        0,0,0,0,c.G,
        0,0,0,0,c.B,
```

```
        0,0,0,1,0
    });
t.View.Background.SetColorFilter(CM);
t.Show();
```

А также можно изменить цвет текста, если фон светлый или темный:

```
if ((((float)(c.R) + (float)(c.G) + (float)(c.B)) / 3) >= 128)
    t.View.FindViewById<TextView>(Android.Resource.Id.Message).SetTextColor(Color.Black);
else
//text color is white by default
```

## Изменить положение тоста

Мы можем изменить наш тост, используя метод SetGravity. Этот метод принимает три параметра: во-первых, гравитация тоста на экране, а две другие устанавливают тост, смещенный от исходного положения (которое задается первым параметром):

```
//Toast at bottom left corner of screen
Toast t = Toast.MakeText(context, message, duration);
t.SetGravity(GravityFlags.Bottom | GravityFlags.Left, 0, 0);
t.Show();

//Toast at a custom position on screen
Toast t = Toast.MakeText(context, message, duration);
t.SetGravity(GravityFlags.Top | GravityFlags.Left, x, y);
t.Show();
```

Прочитайте Тосты онлайн: https://riptutorial.com/ru/xamarin-android/topic/3550/тосты

# кредиты

| S. No | Главы | Contributors |
|---|---|---|
| 1 | Начало работы с Xamarin.Android | Amy Burns, Community, Jon Douglas, Kevin Montrose, Ryan Weaver |
| 2 | RecyclerView | Alexandre, Matthew, Ryan Alford, Sreeraj, Zverev Eugene |
| 3 | Xamarin.Android - Bluetooth-связь | Ladislav |
| 4 | Xamarin.Android - Как создать панель инструментов | Daniel Krzyczkowski, tylerjgarland |
| 5 | Диалоги | JimBobBennett, Pilatus |
| 6 | Жизненный цикл приложения - Xamarin.Andorid | CDrosos, Daniel Krzyczkowski, Steven Mark Ford |
| 7 | Как исправить ориентацию изображения, снятого с устройства Android | Daniel Krzyczkowski |
| 8 | Наручники | EJoshuaS, Jon Douglas, jonp, Matthew, Prashant C, Sven-Michael Stübe |
| 9 | Пользовательский ListView | user3814750 |
| 10 | Публикация вашего Xamarin.Android APK | Alexandre |
| 11 | Сканирование штрих-кода с использованием библиотеки ZXing в приложениях Xamarin | GvSharma |

| | | |
|---|---|---|
| 12 | Тосты | GONeale, Matthew, Piet, user2912553 |