eBook Gratuit

APPRENEZ Xamarin.Android

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#xamarin.an

droid

Table des matières

À propos

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

Chapitre 1: Démarrer avec Xamarin.Android

Remarques

Xamarin.Android vous permet de créer des applications Android natives en utilisant les mêmes contrôles de l'interface utilisateur que vous le feriez en Java, sauf avec la flexibilité et l'élégance d'un langage moderne (C #), la puissance de la BCL (Base Class Library). IDE de première classe - Xamarin Studio et Visual Studio - à portée de main.

Pour plus d'informations sur l'installation de Xamarin.Android sur votre ordinateur Mac ou Windows, consultez les guides de démarrage du centre de développement Xamarin.

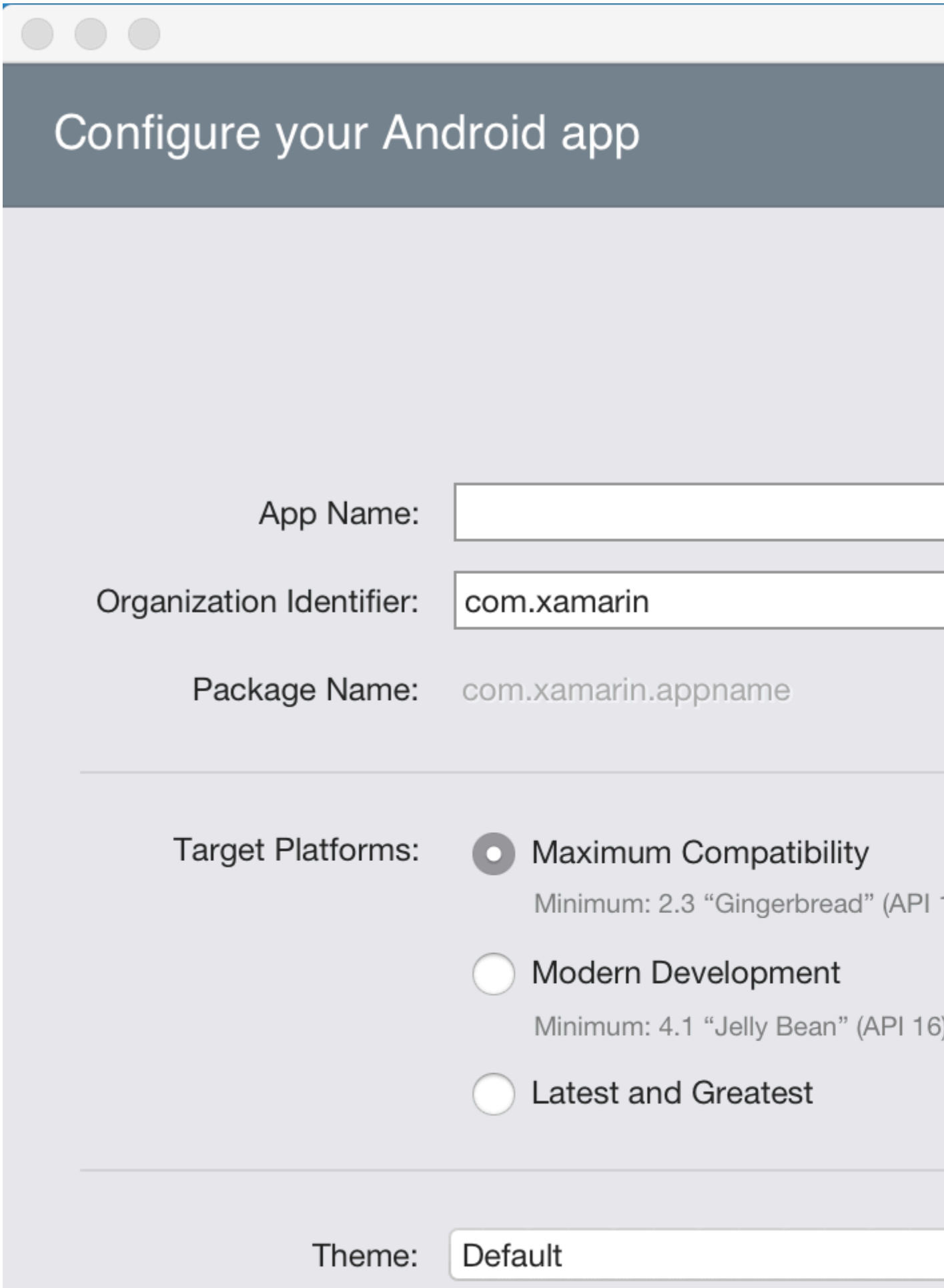
Versions

Version	Nom de code	Niveau API	Date de sortie
1.0	Aucun	1	2008-09-23
1.1	Aucun	2	2009-02-09
1,5	Petit gâteau	3	2009-04-27
1.6	Beignet	4	2009-09-15
2.0-2.1	Eclair	5-7	2009-10-26
2.2-2.2.3	Froyo	8	2010-05-20
2.3-2.3.7	pain d'épice	9-10	2010-12-06
3.0-3.2.6	Rayon de miel	11-13	2011-02-22
4.0-4.0.4	Sandwich à la crème glacée	14-15	2011-10-18
4.1-4.3.1	Dragée	16-18	2012-07-09
4.4-4.4.4, 4.4W-4.4W.2	KitKat	19-20	2013-10-31
5.0-5.1.1	Sucette	21-22	2014-11-12
6.0-6.0.1	Guimauve	23	2015-10-05
7.0	Nougat	24	2016-08-22

Examples

Commencez avec Xamarin Studio

1. Accédez à **Fichier> Nouveau> Solution** pour afficher la nouvelle boîte de dialogue de projet.
2. Sélectionnez **Android App** et appuyez sur **Suivant** .
3. Configurez votre application en définissant le nom de votre application et l'ID de l'organisation. Sélectionnez la plate-forme cible la mieux adaptée à vos besoins ou laissez-la par défaut. Appuyez sur Suivant:

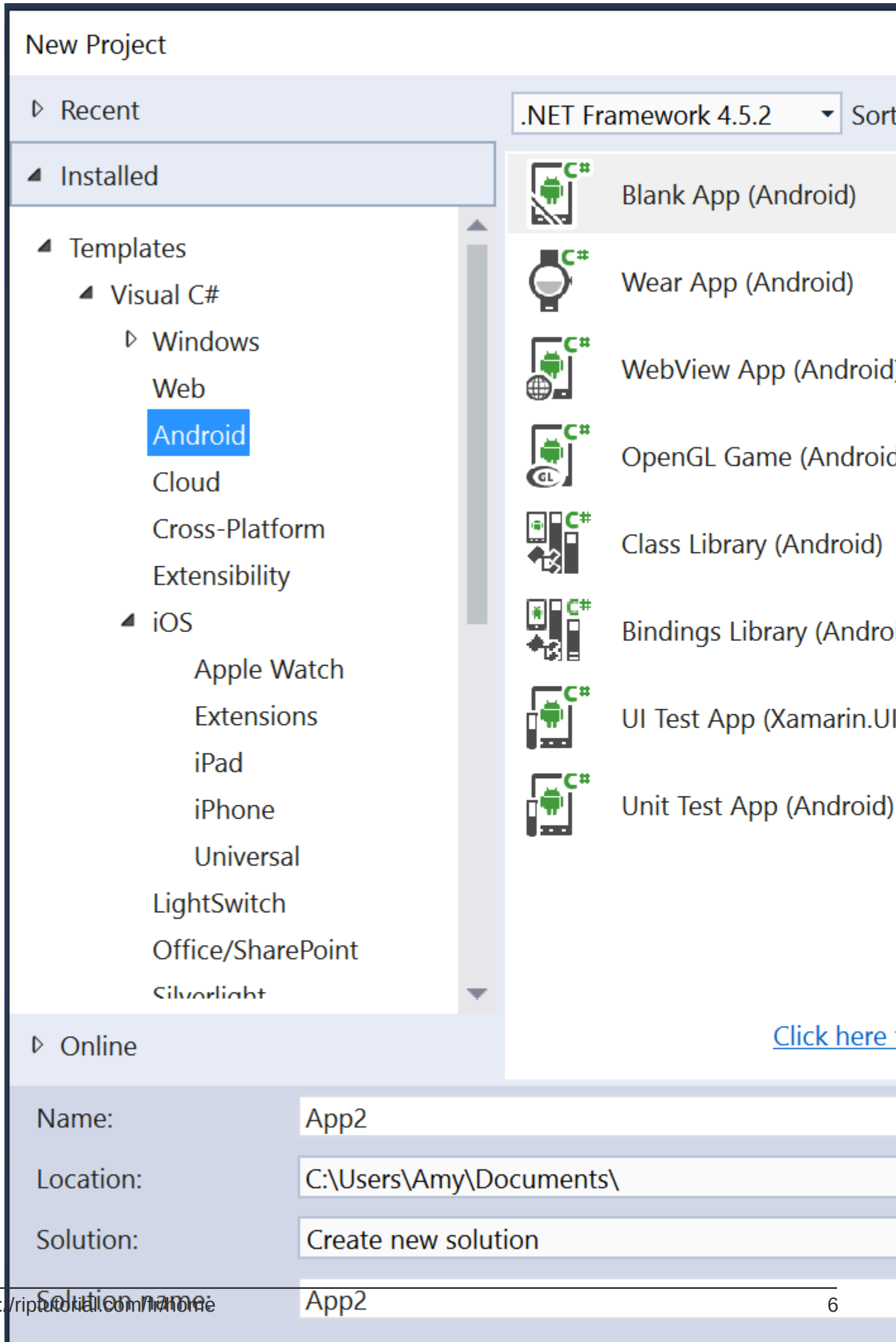

Configure your Android app
App Name:
Organization Identifier:
com.xamarin
Package Name:
com.xamarin.appname
Target Platforms:
Maximum Compatibility
Minimum: 2.3 “Gingerbread” (API
Modern Development
Minimum: 4.1 “Jelly Bean” (API 16)
Latest and Greatest
Theme:
Default

4. Définissez le nom de votre projet et le nom de la solution, ou laissez-le comme nom par défaut. Cliquez sur Créer pour créer votre projet.
5. Configurer votre appareil pour le déploiement ou configurer un émulateur
6. Pour exécuter votre application, sélectionnez la configuration de **débogage** et appuyez sur le bouton de lecture:

   et appuyez sur le bouton de lecture:

Commencer dans Visual Studio

1. Accédez à **Fichier> Nouveau> Projet** pour afficher la boîte de dialogue Nouveau projet.
2. Naviguez vers **Visual C #> Android** et sélectionnez 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
Online
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)
Click here
Name:
App2
Location:
C:\Users\Amy\Documents\
Solution:
Create new solution
Solution name:
App2

3. Donnez un **nom à** votre application et appuyez sur **OK** pour créer votre projet.
4. Configurer votre appareil pour le déploiement ou configurer un émulateur
5. Pour exécuter votre application, sélectionnez la configuration de **débogage** et appuyez sur

   le bouton **Démarrer** :
   et appuyez sur le bouton **Démarrer** :

Lire Démarrer avec Xamarin.Android en ligne: https://riptutorial.com/fr/xamarin-android/topic/403/demarrer-avec-xamarin-android

Chapitre 2: Balayage de codes à barres à l'aide de la bibliothèque ZXing dans les applications Xamarin

Introduction

La bibliothèque Zxing est bien connue pour le traitement des images. Zxing était basé sur Java et le module .Net est également disponible et il peut être utilisé dans les applications xamarin. cliquez ici pour vérifier la documentation officielle. http://zxingnet.codeplex.com/

J'ai récemment utilisé cette librairie.

Etape 1: Ajouter le composant ZXing.Net.Mobile dans la solution.

step2: Quelle que soit l'activité dont nous avons besoin pour afficher le lecteur de code à barres, dans cette activité, initialisez MobileBarcodeScanner.

Etape 3: Ecrivez le code ci-dessous lorsque vous appuyez sur n'importe quelle vue pour lancer la numérisation.

Examples

Exemple de code

```
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;
}
```

Lire Balayage de codes à barres à l'aide de la bibliothèque ZXing dans les applications Xamarin en ligne: https://riptutorial.com/fr/xamarin-android/topic/9526/balayage-de-codes-a-barres-a-l-aide-de-la-bibliotheque-zxing-dans-les-applications-xamarin

Chapitre 3: Comment corriger l'orientation d'une image capturée depuis un appareil Android

Remarques

1. Cet exemple d'application est disponible sur mon GitHub ci-dessous:

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

2. La documentation du composant Xamarin Mobile est disponible ci-dessous:

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

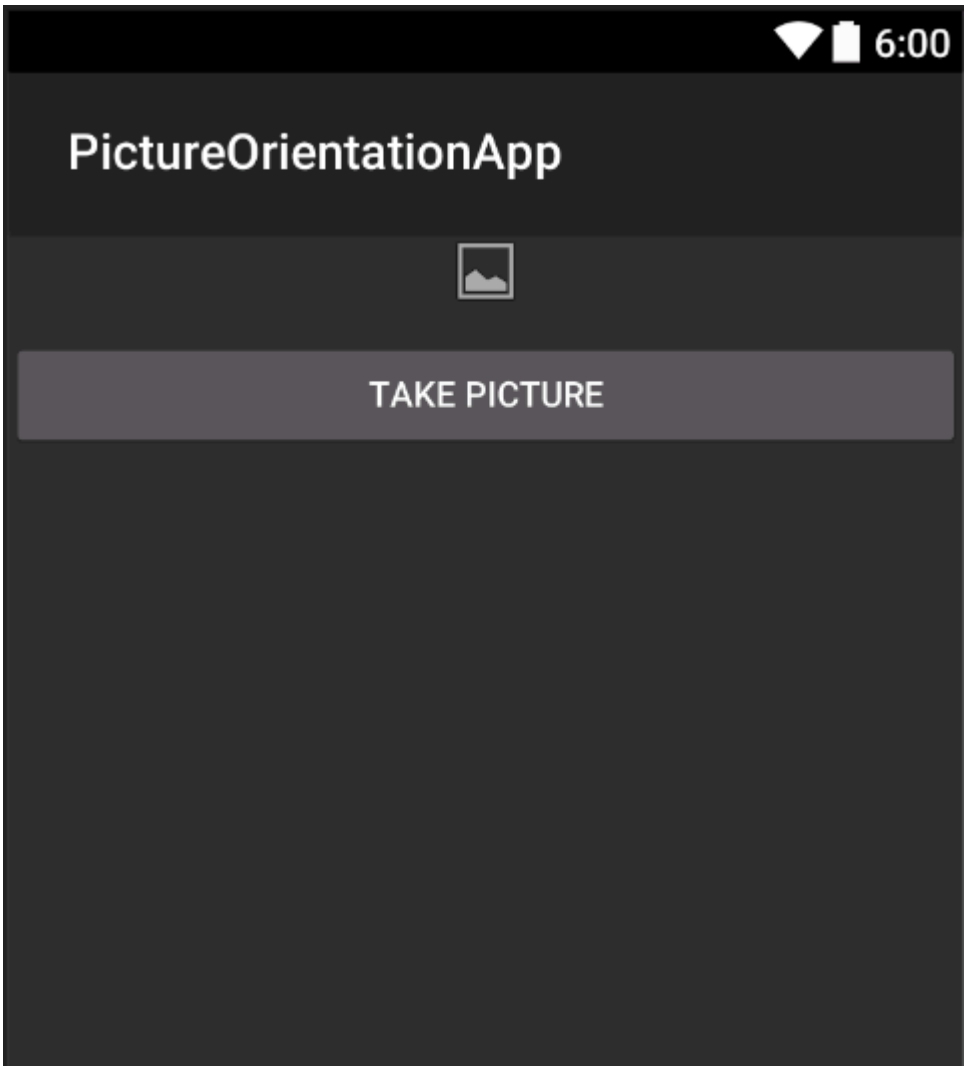
Examples

Comment corriger l'orientation d'une image capturée depuis un appareil Android

Cet exemple montre comment prendre une image et l'afficher correctement sur le périphérique Android.

Tout d'abord, nous devons créer un exemple d'application avec un bouton et une image. Une fois que l'utilisateur clique sur le bouton, la caméra est lancée et, une fois que l'utilisateur sélectionne l'image, celle-ci s'affiche avec l'orientation appropriée sur l'écran.

1. Ajouter un bouton nommé "TakePictureButton" et imageview nommé "TakenPictureImageView":

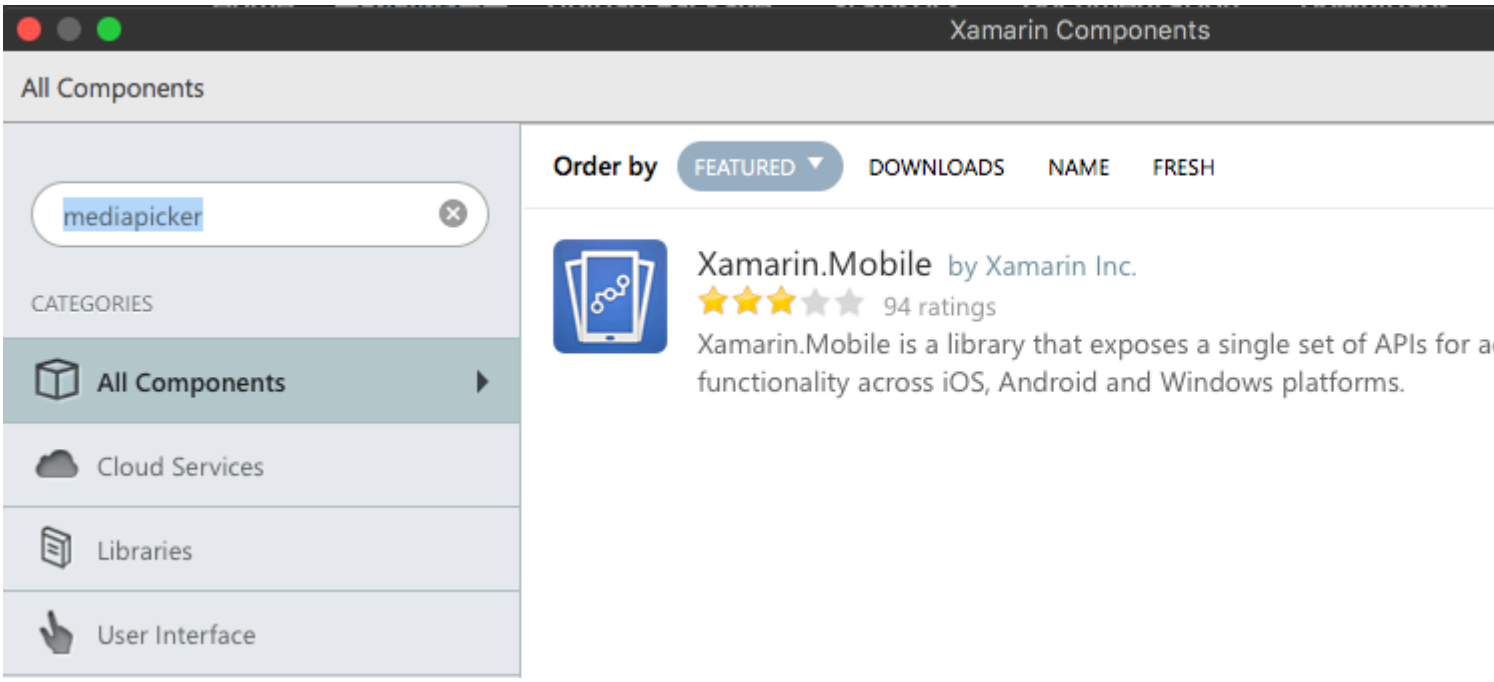


2. Maintenant, ouvrez le code d'activité derrière:

Ici, premièrement, faites référence à vos commandes:

```
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. Dans notre application, nous utiliserons le composant Xamarin Mobile disponible dans le magasin de composants:



4. Une fois que vous l'ajoutez au projet, nous pouvons continuer. Ajouter ci-dessous le code qui est responsable du lancement de la caméra. Cette méthode doit être appelée dans le clic du bouton comme vous pouvez le voir dans le code ci-dessus:

```
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. Une fois que l'utilisateur prend la photo, nous devons l'afficher dans la bonne orientation. Pour le faire, utilisez la méthode ci-dessous. Il est chargé de récupérer les informations d'exif de l'image prise (y compris l'orientation au moment de la prise de vue) et de créer une image bitmap avec la bonne orientation:

```
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;
```

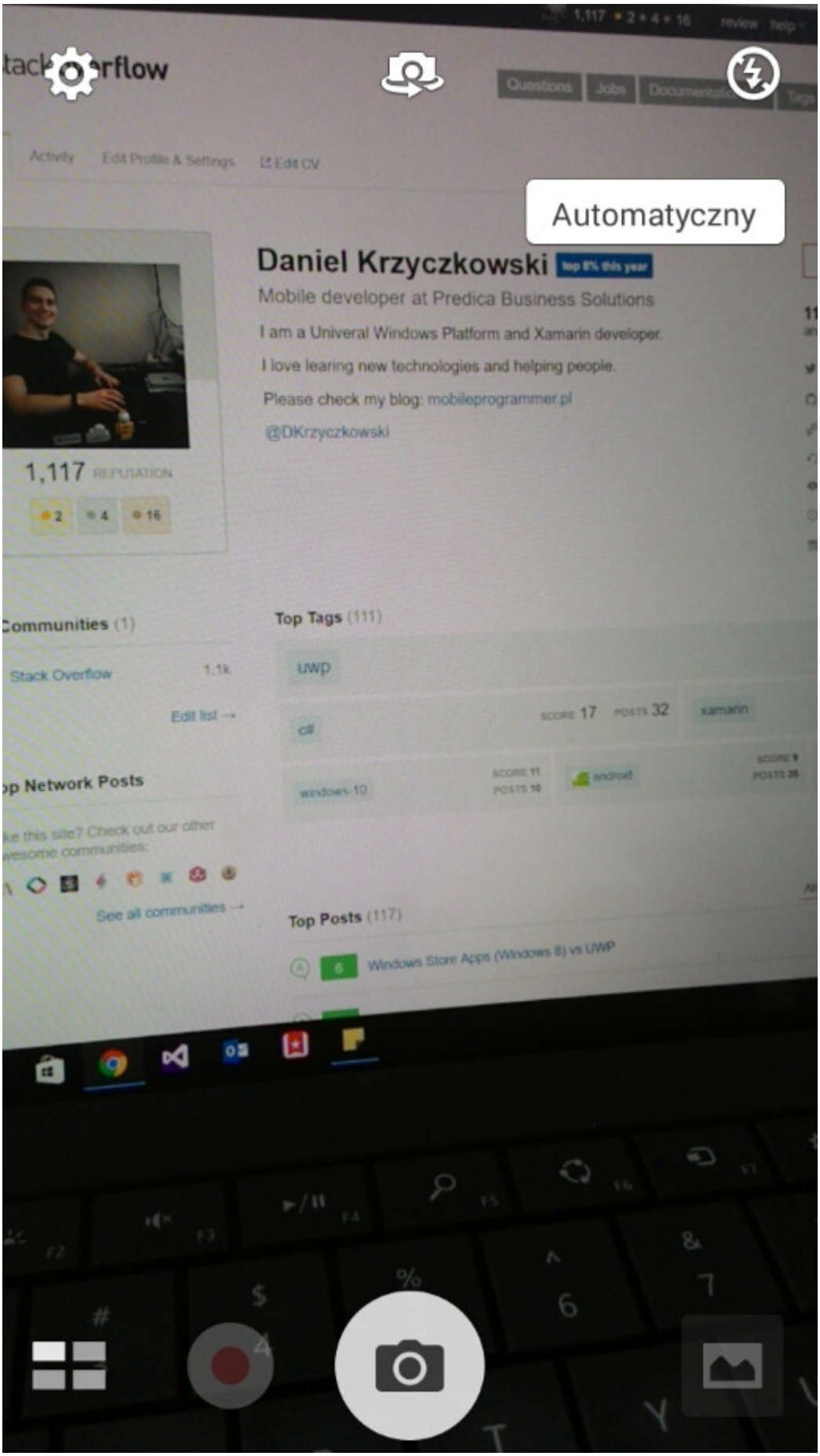
```
        height_tmp /= 2;
        scale++;
    }

    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;
```

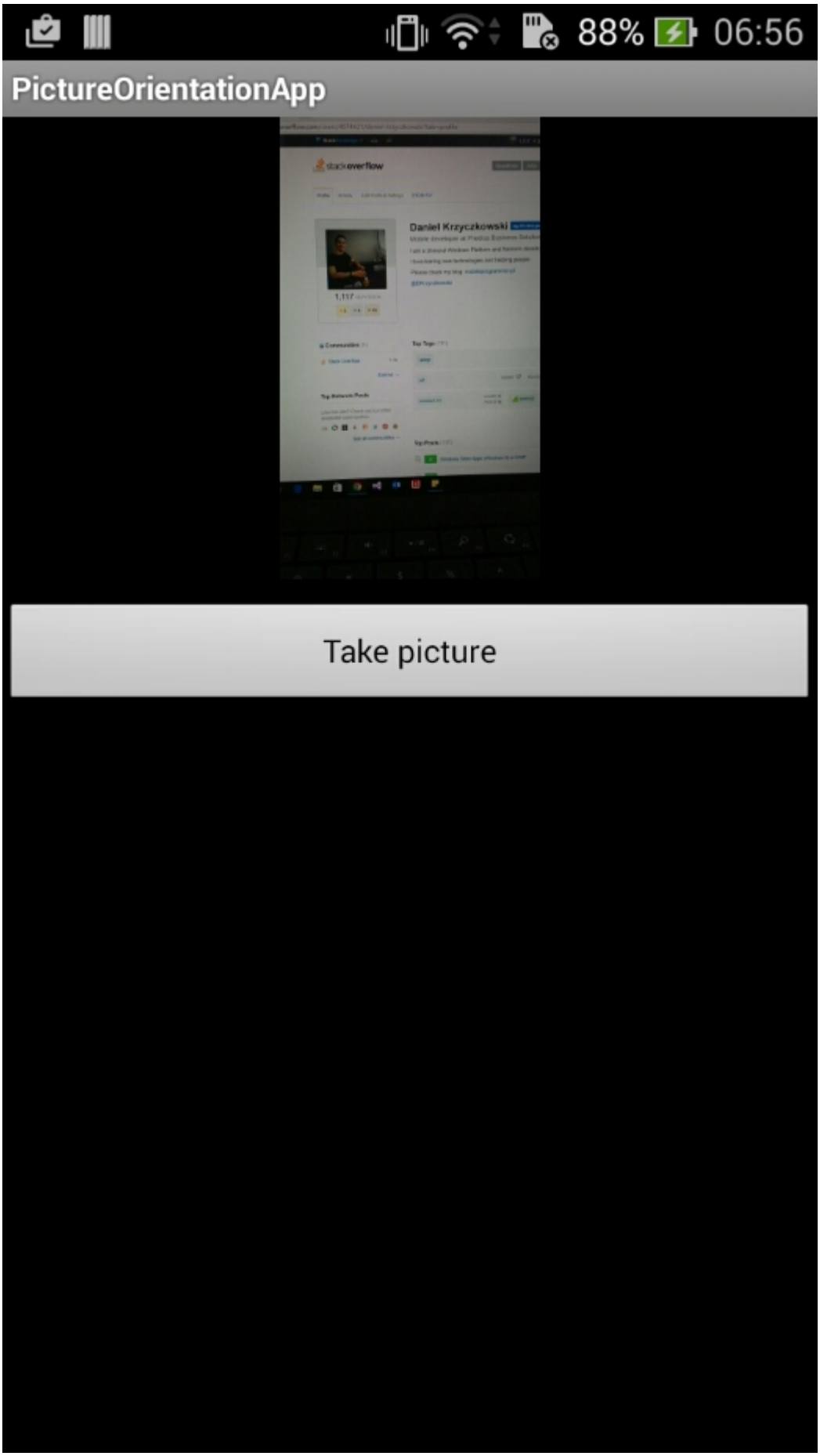
```
        }

        return resizedBitmap;
    }

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

6. La méthode ci-dessus doit être appelée dans la méthode OnActivityResult appelée après que l'utilisateur a pris la photo:

```
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. Lancez l'application. Prendre une photo et voir le résultat:


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
Top Tags (111)
uwp
c#
xamarin
windows-10
android
Top Posts (117)
Windows Store Apps (Windows 8) vs UWP



C'est tout. Vous allez maintenant avoir toute photo prise dans la bonne orientation.

Lire Comment corriger l'orientation d'une image capturée depuis un appareil Android en ligne: https://riptutorial.com/fr/xamarin-android/topic/6683/comment-corriger-l-orientation-d-une-image-capturee-depuis-un-appareil-android

Chapitre 4: Cycle de vie de l'application - Xamarin.Andorid

Introduction

Le cycle de vie de l'application Xamarin.Android est identique à l'application Android normale. Lorsque nous parlons de cycle de vie, nous devons parler du cycle de vie des applications, du cycle de vie des activités et du cycle de vie des fragments.

Dans ce qui suit, je vais essayer de fournir une bonne description et une bonne façon de les utiliser. J'ai obtenu cette documentation à partir de la documentation officielle Android et Xamarin et de nombreuses ressources Web utiles fournies dans la section Remarques ci-dessous.

Remarques

Liens intéressants pour élargir vos connaissances sur le cycle de vie d'une application 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

Cycle de vie de l'application

Tout d'abord, vous devez savoir que vous pouvez étendre la classe Android.Application pour pouvoir accéder à deux méthodes importantes liées au cycle de vie des applications:

- OnCreate - Appelé au démarrage de l'application, avant la création de tout autre objet d'application (comme MainActivity).
- OnTerminate - Cette méthode est destinée aux environnements de processus émulés. Il ne sera jamais appelé sur un appareil Android de production, où les processus sont supprimés simplement en les tuant; Aucun code utilisateur (y compris ce rappel) n'est exécuté lors de cette opération. De la documentation: https://developer.android.com/reference/android/app/Application.html#onTerminate ()

Dans l'application Xamarin.Android, vous pouvez étendre la classe Application de la manière présentée ci-dessous. Ajoutez une nouvelle classe appelée "MyApplication.cs" à votre projet:

```
[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();
    }
}
```

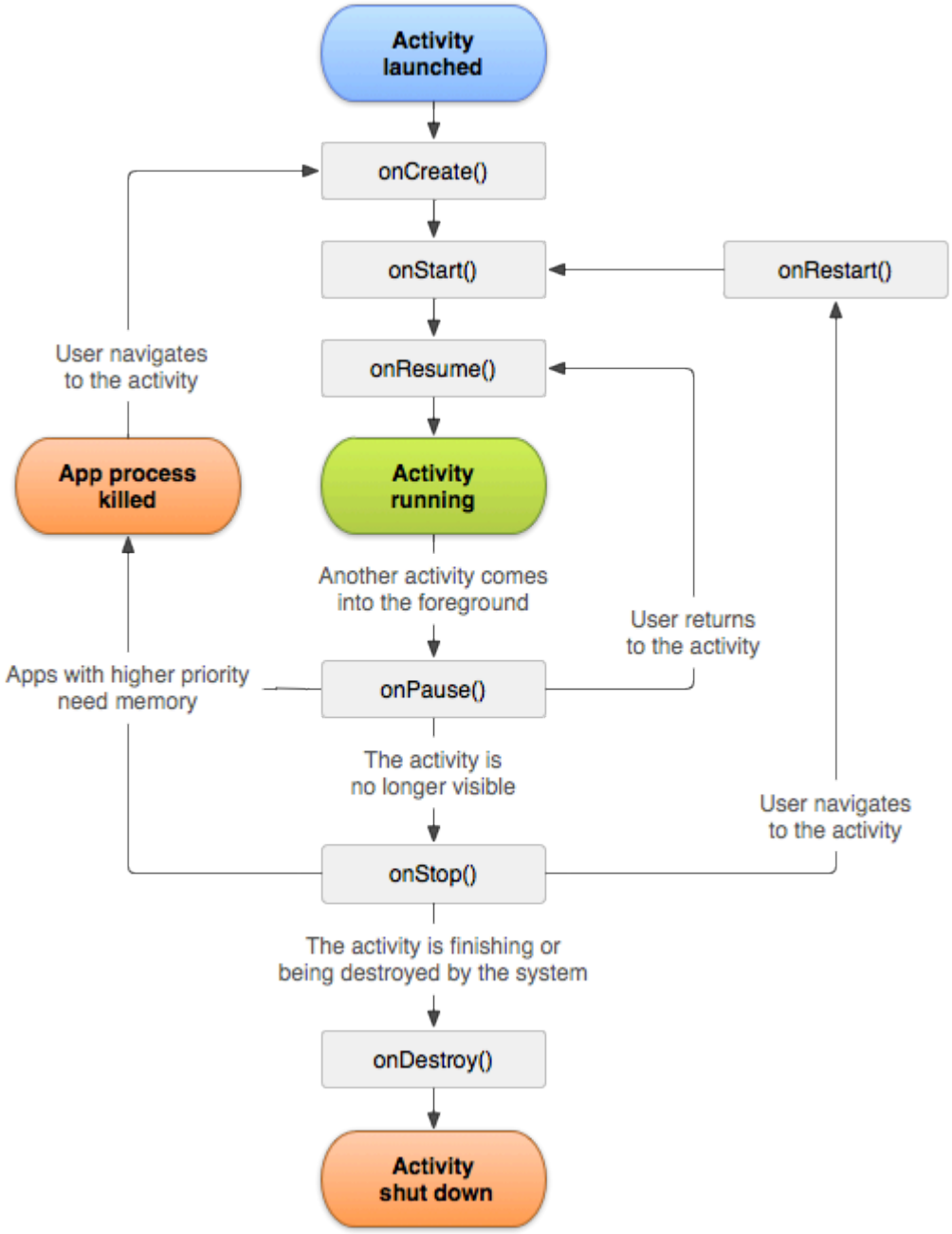
Comme vous l'avez écrit ci-dessus, vous pouvez utiliser la méthode OnCreate. Vous pouvez par exemple initialiser la base de données locale ici ou configurer une configuration supplémentaire.

Il existe également d'autres méthodes qui peuvent être remplacées comme: OnConfigurationChanged ou OnLowMemory.

Cycle de vie de l'activité

Le cycle de vie de l'activité est beaucoup plus complexe. Comme vous le savez, l'activité est une seule page dans l'application Android où l'utilisateur peut interagir avec elle.

Sur le diagramme ci-dessous, vous pouvez voir à quoi ressemble le cycle de vie d'une activité Android:



Comme vous pouvez le constater, il existe un flux spécifique de cycle de vie d'activité. Dans l'application mobile, vous avez bien sûr des méthodes dans chaque classe d'activité qui gèrent un fragment de cycle de vie spécifique:

```
[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");
    }
}
```

Il y a une bonne description dans la documentation officielle d'Android:

- La durée de vie totale d'une activité se produit entre le premier appel à onCreate (Bundle) et un seul appel final à onDestroy (). Une activité effectuera toutes les configurations de l'état "global" dans onCreate () et libèrera toutes les ressources restantes dans onDestroy (). Par exemple, si un thread s'exécute en arrière-plan pour télécharger des données du réseau, il peut créer ce thread dans onCreate (), puis arrêter le thread dans onDestroy ().

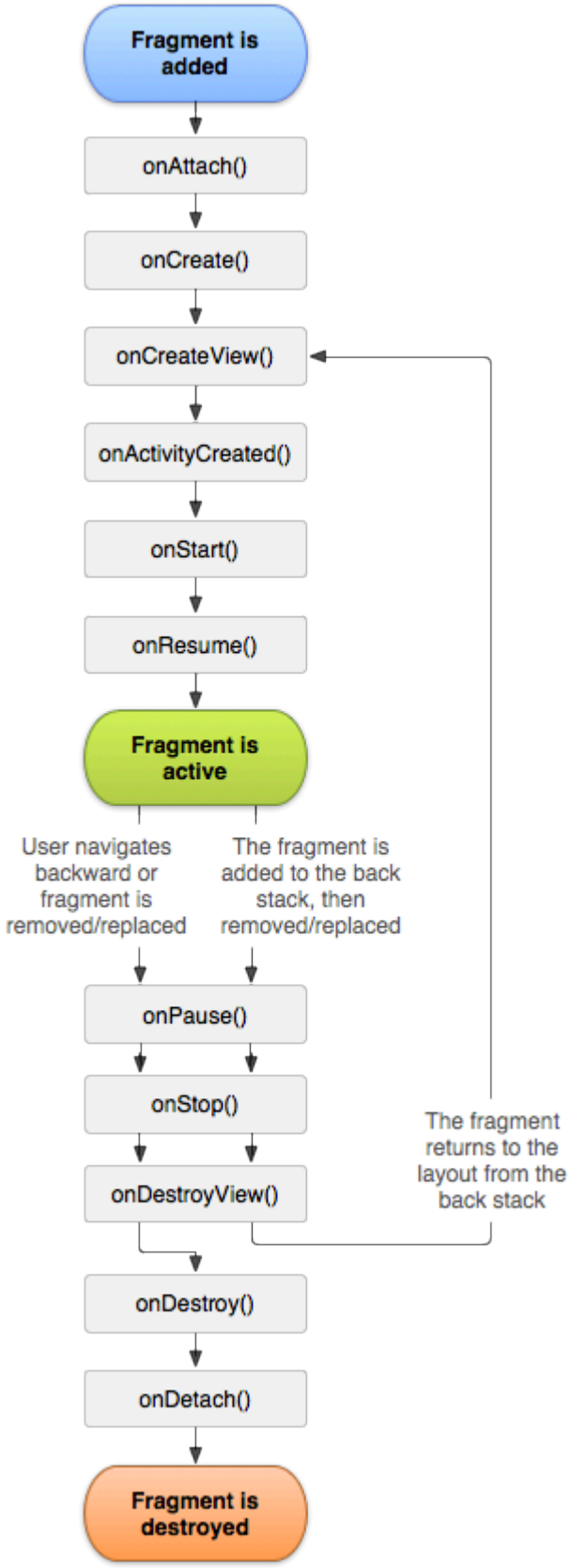
- La durée de vie visible d'une activité se produit entre un appel à onStart () et un appel correspondant à onStop (). Pendant ce temps, l'utilisateur peut voir l'activité à l'écran, bien qu'il puisse ne pas être au premier plan et interagir avec l'utilisateur. Entre ces deux méthodes, vous pouvez gérer les ressources nécessaires pour afficher l'activité à l'utilisateur. Par exemple, vous pouvez enregistrer un BroadcastReceiver dans onStart () pour surveiller les modifications qui ont un impact sur votre interface utilisateur, puis annuler son enregistrement dans onStop () lorsque l'utilisateur ne voit plus ce que vous affichez. Les méthodes onStart () et onStop () peuvent être appelées plusieurs fois, à mesure que l'activité devient visible et masquée pour l'utilisateur.

- La durée de vie au premier plan d'une activité se produit entre un appel à onResume () et un appel correspondant à onPause (). Pendant ce temps, l'activité est devant toutes les autres activités et interagit avec l'utilisateur. Une activité peut souvent aller entre les états repris et mis en pause - par exemple lorsque le périphérique se met en veille, lorsqu'un résultat d'activité est délivré, qu'une nouvelle intention est délivrée - le code de ces méthodes doit donc être relativement léger.

Cycle de vie des fragments

Comme vous le savez, vous pouvez avoir une activité mais des fragments différents intégrés. C'est pourquoi le cycle de vie des fragments est également important pour les développeurs.

Sur le diagramme ci-dessous, vous pouvez voir à quoi ressemble le cycle de vie d'un fragment Android:



Comme décrit dans la documentation officielle d'Android, vous devez implémenter au moins trois méthodes:

- OnCreate - le système appelle cela lors de la création du fragment. Dans votre implémentation, vous devez initialiser les composants essentiels du fragment que vous souhaitez conserver lorsque le fragment est suspendu ou arrêté, puis repris.

- OnCreateView - Le système appelle cela lorsqu'il est temps que le fragment dessine son interface utilisateur pour la première fois. Pour dessiner une interface utilisateur pour votre fragment, vous devez renvoyer une vue de cette méthode qui est la racine de la présentation de votre fragment. Vous pouvez retourner null si le fragment ne fournit pas d'interface utilisateur.

- OnPause - Le système appelle cette méthode comme première indication que l'utilisateur quitte le fragment (bien que cela ne signifie pas toujours que le fragment est détruit). C'est généralement là que vous devez commettre les modifications qui doivent être persistées au-delà de la session utilisateur en cours (car l'utilisateur risque de ne pas revenir).

Voici un exemple d'implémentation dans 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();
    }
}
```

Bien sûr, vous pouvez ajouter des méthodes supplémentaires si vous souhaitez gérer différents états.

Exemple complet sur GitHub

Si vous souhaitez obtenir un projet de base avec les méthodes décrites ci-dessous, vous pouvez télécharger le modèle d'application Xamarin.Android à partir de mon GitHub. Vous pouvez trouver des exemples pour:

- Méthodes de cycle de vie des applications

- Méthodes de cycle de vie d'activité
- Méthodes de cycle de vie des fragments

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

Lire Cycle de vie de l'application - Xamarin.Andorid en ligne: https://riptutorial.com/fr/xamarin-android/topic/8842/cycle-de-vie-de-l-application---xamarin-andorid

Chapitre 5: Dialogues

Remarques

Définir le `Context` de la boîte de dialogue

Lors de la création d' une `Dialog` de `Activiy` `this` `Dialog` à partir d' un `Activiy` nous pouvons utiliser `this` que le contexte.

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

Avec `Fragments` nous utilisons la propriété `Context` .

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

Types de boutons

`SetNeutralButton()` peut être utilisé pour une simple notification et la confirmation que la notification est lue. `SetPositiveButton()` peut être utilisé pour une confirmation par exemple: "Êtes-vous sûr de vouloir supprimer cet élément?" `SetNegativeButton()` sert à `SetNegativeButton()` la boîte de dialogue et à annuler son action.

Désactiver l'annulation du bouton retour

Si nous voulons nous assurer que l'utilisateur ne peut pas fermer la boîte de dialogue avec le bouton Précédent, nous pouvons appeler `SetCanceable(false)` . Cela ne fonctionne que pour le bouton retour.

Rotation

Si l'écran est pivoté pendant qu'une boîte de dialogue est visible, il sera supprimé et les actions ok et cancel ne seront pas appelées. Vous devrez gérer cela dans votre activité et afficher à nouveau la boîte de dialogue une fois l'activité rechargée.

Pour contourner ce `DialogFragment` plutôt un `DialogFragment` .

Examples

Boîte de dialogue d'alerte

Création d'une boîte de dialogue d'alerte

```
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();
```

Lire Dialogues en ligne: https://riptutorial.com/fr/xamarin-android/topic/2510/dialogues

Chapitre 6: Dialogues

Paramètres

méthode publique couramment utilisée	Utilisation
SetTitle (String)	Définit le titre de la boîte de dialogue
SetIcon (dessinable)	Définir l'icône pour la boîte de dialogue d'alerte
SetMessage (chaîne)	Définissez le message à afficher.
SetNegativeButton (String, EventHandler)	Définissez un écouteur à appeler lorsque vous appuyez sur le bouton négatif de la boîte de dialogue.
SetPositiveButton (String, EventHandler)	Définissez un écouteur à appeler lorsque vous appuyez sur le bouton positif de la boîte de dialogue.
SetNeutralButton (String, EventHandler)	Définissez un écouteur à appeler lorsque vous appuyez sur le bouton neutre de la boîte de dialogue.
SetOnCancelListener (IDialogInterfaceOnCancelListener)	Définit le rappel qui sera appelé si la boîte de dialogue est annulée.
SetOnDismissListener (IDialogInterfaceOnDismissListener)	Définit le rappel qui sera appelé lorsque la boîte de dialogue est rejetée pour une raison quelconque.
Montrer()	Crée un AlertDialog avec les arguments fournis à ce générateur et la boîte de dialogue Dialog.Show.

Remarques

Exigences

Espace de noms: Android.App

Assembly: Mono.Android (dans Mono.Android.dll)

Versions d'assemblage: 0.0.0.0

Constructeurs Publics

AlertDialog.Builder (Contexte): -

Constructeur utilisant un contexte pour ce générateur et le AlertDialog qu'il crée.

AlertDialog.Builder (Contexte, Int32): -

Constructeur utilisant un contexte et un thème pour ce générateur et le AlertDialog qu'il crée.

Utiliser AlertDialog de conception de matériaux

Pour utiliser le AlertDialog moderne:

1. Installer la bibliothèque AppCompat de Support v7 à partir des packages NuGet
2. Remplacez AlertDialog par Android.Support.V7.App.AlertDialog ou ajoutez la déclaration suivante en haut pour que votre boîte de dialogue brille.

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

Examples

AlertDialog

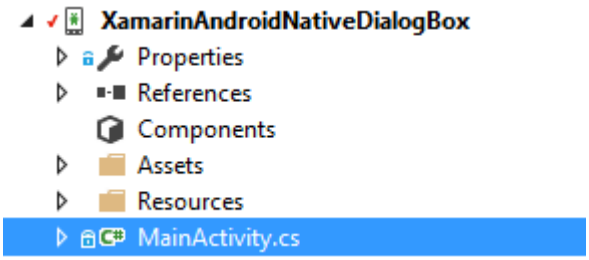
```
// 1. Instantiate an AlertDialog.Builder with its constructor
// the parameter this is the context (usually your activity)
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// 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();
```

Exemple de dialogue d'alerte simple

Nous allons créer un simple dialogue d'alerte dans Xamarin.Android

Maintenant, considérant que vous avez parcouru le guide de démarrage de la documentation.

Vous devez avoir la structure du projet comme ceci:



Votre activité principale doit ressembler à ceci:

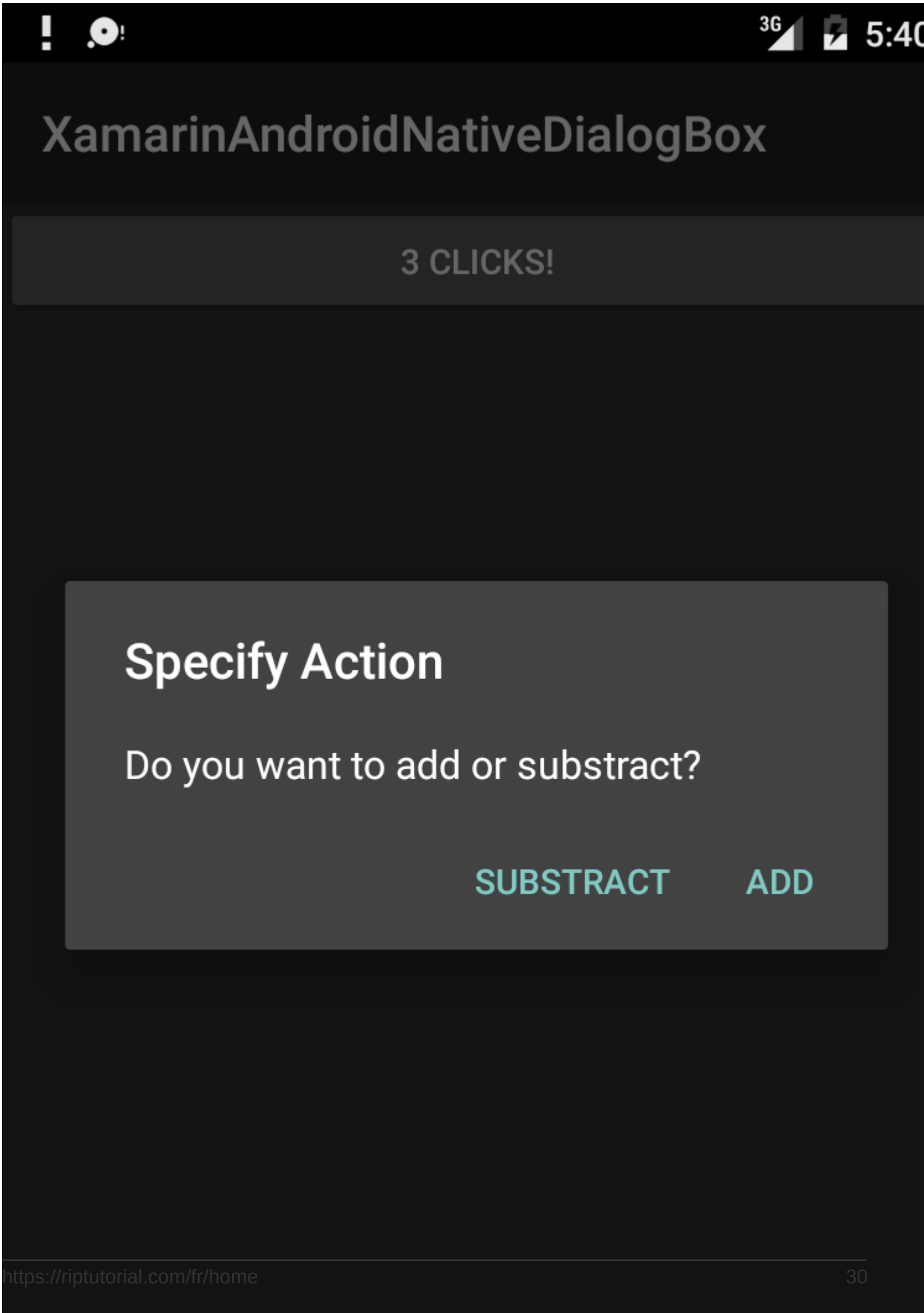
```
public class MainActivity : Activity
{
int count = 1;

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++); };
}
}
```

Maintenant, ce que nous allons faire, au lieu d'ajouter un au compteur sur le clic du bouton, nous demanderons à l'utilisateur s'il veut ajouter ou soustraire un dans un simple dialogue d'alerte.

Et sur le clic du bouton positif ou négatif, nous prendrons l’action.

```
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();
};
```

capture d'écran:


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

Chapitre 7: Fixations

Examples

Suppression de types

Il est possible d'indiquer au générateur Xamarin.Android Bindings d'ignorer un type Java et de ne pas le lier. Cela se fait en ajoutant un élément XML `remove-node` au fichier metadata.xml:

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

Implémentation des interfaces Java

Si une bibliothèque Java contient des interfaces qui doivent être implémentées par l'utilisateur (par exemple, des écouteurs de clic comme `View.IOnClickListener` ou des rappels), la classe d'implémentation doit hériter - directement ou indirectement - de `Java.Lang.Object` ou `Java.Lang.Throwable` . C'est une erreur courante, car les étapes de l'emballage ne font qu'imprimer un avertissement qui est facilement ignoré:

> Tapez 'MyListener' implémente Android.Runtime.IJavaObject mais n'hérite pas de Java.Lang.Object. Ce n'est pas supporté.

Faux

L'utilisation de cette implémentation entraînera un comportement inattendu.

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

    public void Dispose()
    {
    }


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

Correct

```
class MyListener :
    Java.Lang.Object, // this is the important part
    View.IOnClickListener
{
    public void OnClick(View v)
    {
        // ...
```

```
    }
}
```

Les bibliothèques de liaisons peuvent renommer les méthodes et les interfaces

Tout dans une bibliothèque de liaisons n'aura pas le même nom en C # qu'en Java.

En C #, les noms d'interface commencent par "I", mais Java n'a pas de convention de ce type. Lorsque vous importez une bibliothèque Java, une interface nommée `SomeInterface` devient `ISomeInterface` .

De même, Java n'a pas de propriétés comme C # le fait. Lorsqu'une bibliothèque est liée, les méthodes Java getter et setter peuvent être modifiées en tant que propriétés. Par exemple, le code Java suivant

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

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

peut être restructuré comme

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

quand c'est lié

Lire Fixations en ligne: https://riptutorial.com/fr/xamarin-android/topic/771/fixations

Chapitre 8: Liste personnalisée

Examples

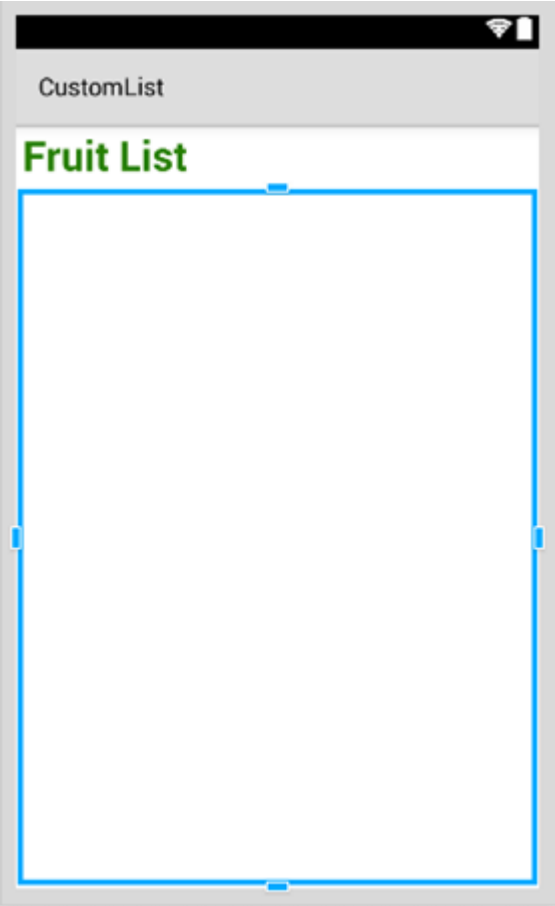
L'aperçu personnalisé comprend des lignes conçues selon les besoins des utilisateurs.

Pour la mise en page au-dessus de votre fichier customrow.axml est comme indiqué ci-dessous

```
    <?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>
```

Ensuite, vous pouvez concevoir votre fichier main.axml, qui contient une vue de texte pour l'en-tête et une vue de liste.



J'espère que c'est facile ...

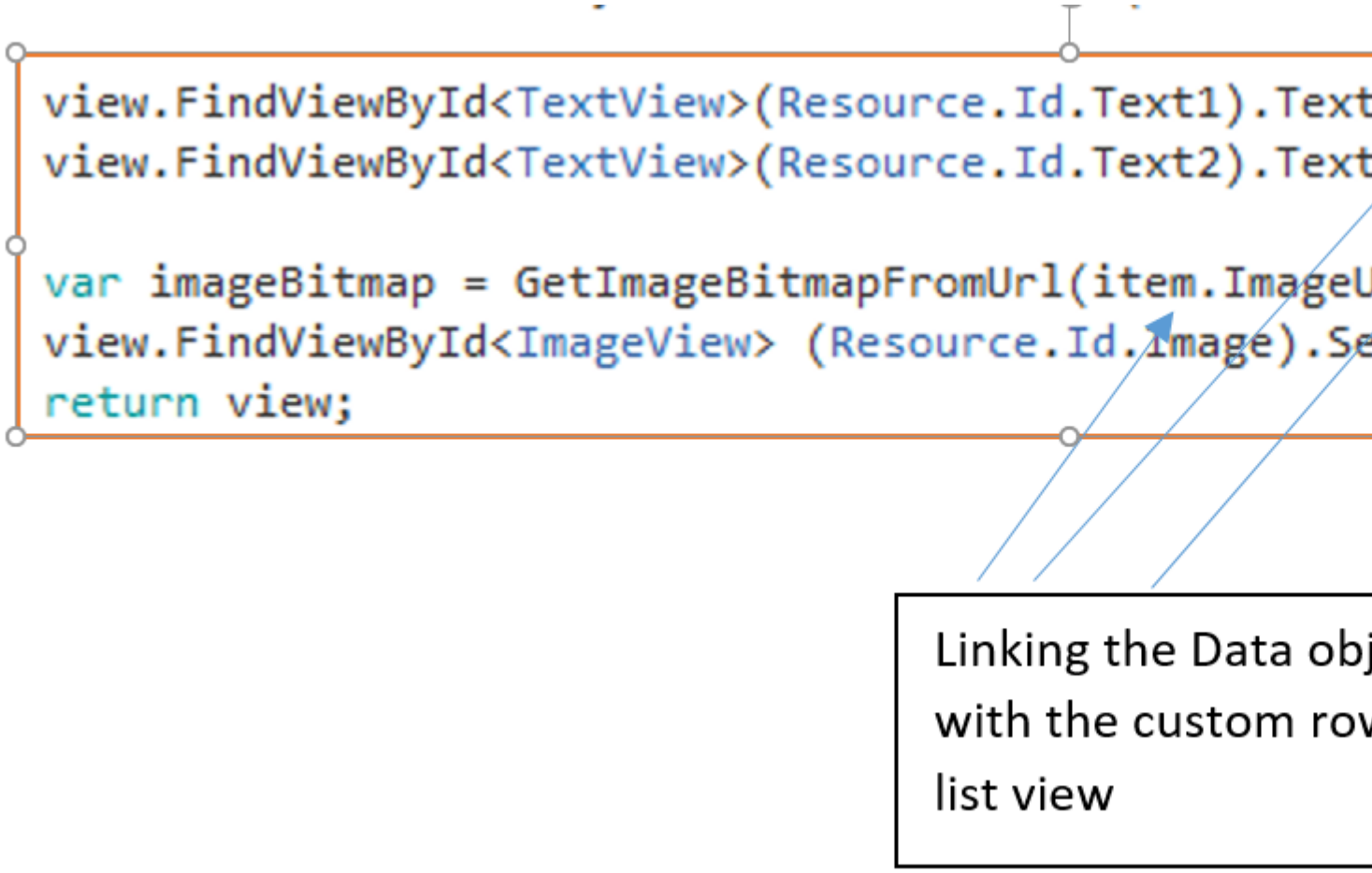
Ensuite, créez votre classe Data.cs qui représentera vos objets de ligne

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

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

Ensuite, vous avez besoin de la classe DataAdapter.cs, les adaptateurs relient vos données à la vue sous-jacente

```
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;
    }

}
```

La partie la plus importante se trouve dans la fonction GetView, c'est là que vous liez votre objet à votre ligne personnalisée.



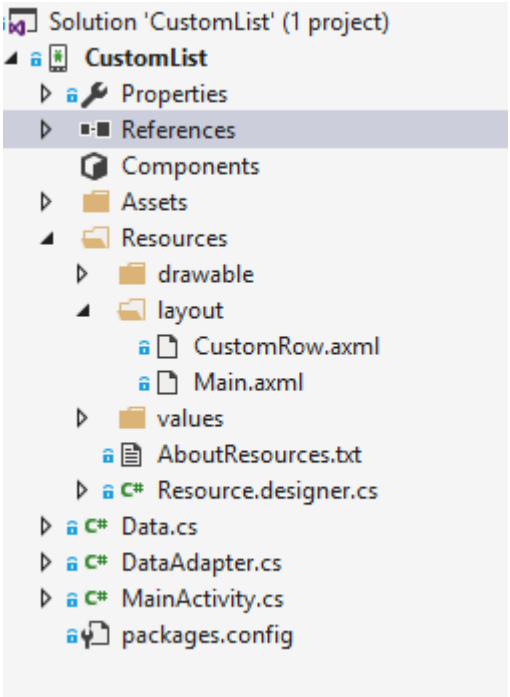
GetImageBitmapFromUrl ne fait pas partie du dataadapter mais je l'ai mis ici pour plus de simplicité.

Enfin, nous arrivons à MainActivity.cs

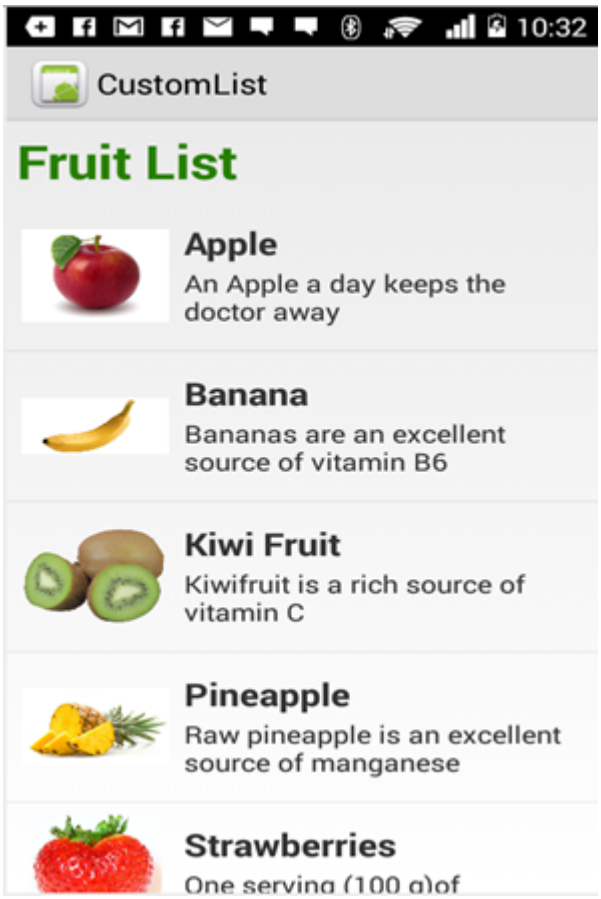
```
public class MainActivity : Activity
{

    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_r


        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);

    }
```

Votre structure de projet finale est comme ci-dessous.



Si tout va bien, vous devriez voir la sortie comme indiqué



Lire Liste personnalisée en ligne: https://riptutorial.com/fr/xamarin-android/topic/6406/liste-personnalisee

Chapitre 9: Publier votre APK Xamarin.Android

Introduction

Cette rubrique affiche des informations sur la préparation de votre application Xamarin.Android pour le mode de publication et son optimisation.

Examples

Préparer votre APK dans Visual Studio

Vous avez terminé votre application, testée en mode débogage et elle fonctionne parfaitement. Maintenant, vous voulez le préparer pour publier dans le Google Play Store.

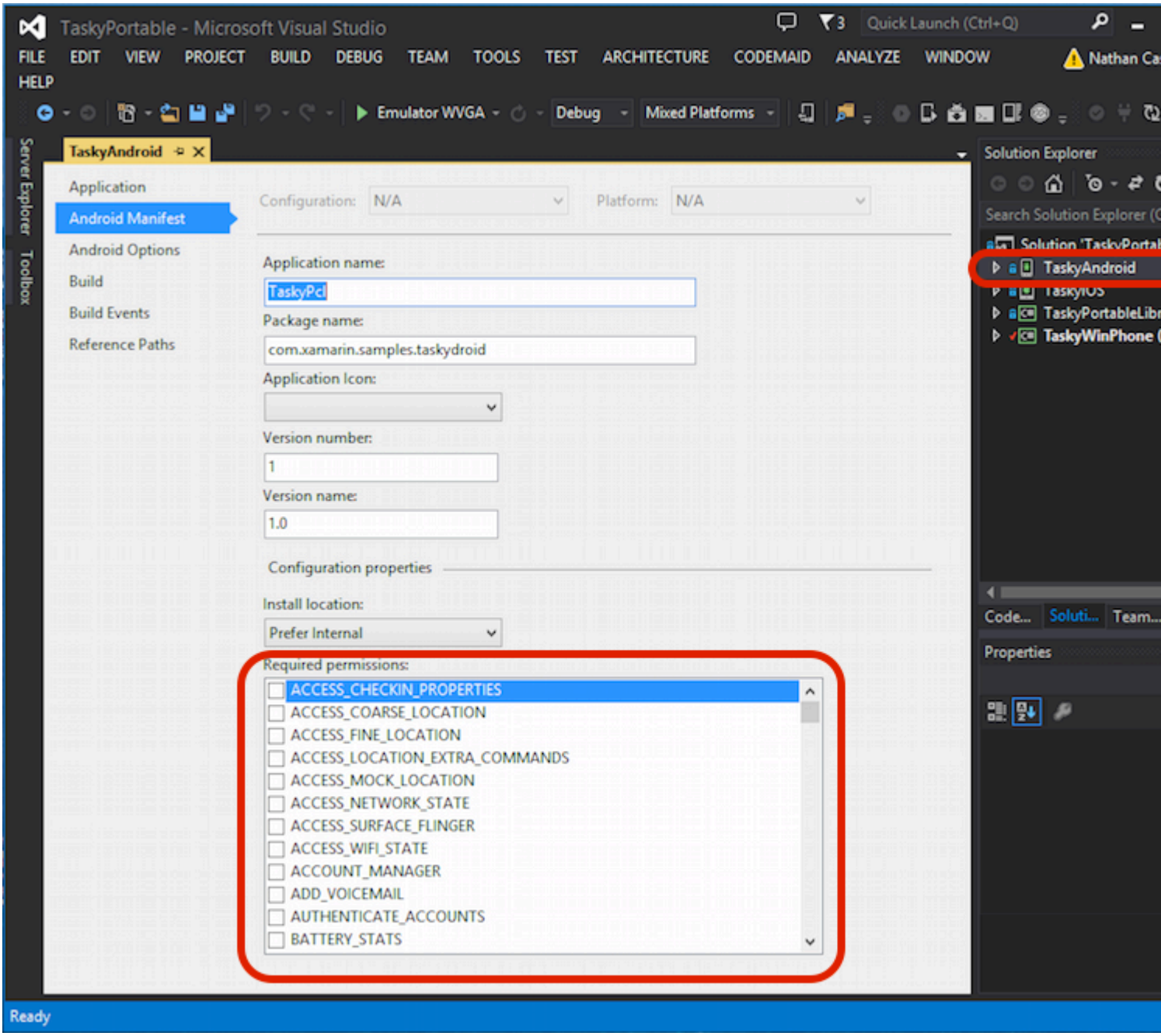
La documentation de Xamarin fournit de bonnes informations ici:

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

---

Manifeste Android

Tout d'abord, dans Visual Studio, cliquez avec le bouton droit sur votre projet Xamarin.Android dans l'Explorateur de solutions et sélectionnez Propriétés. Ensuite, allez dans l'onglet Manifeste Android, pour voir cet écran:



Contrairement à Android Studio ou Eclipse, vous n'avez pas besoin de définir le fichier AndroidManifest.xml en écrivant; Xamarin et Visual Studio font cela pour vous. Activités, BroadcastReceivers et Services sont insérés dans Android Manifest en déclarant des attributs spécifiques dans leurs classes .

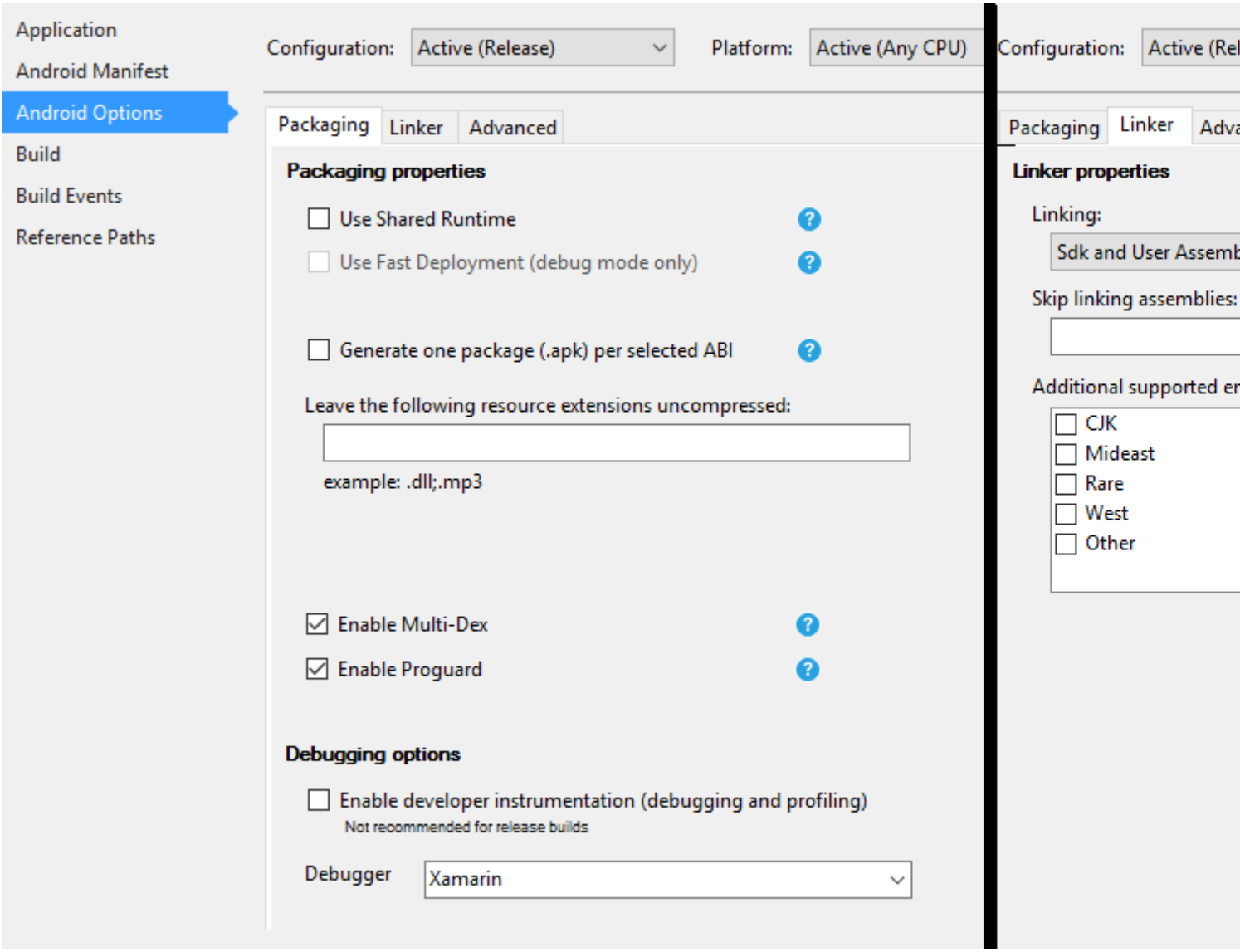
Dans cet écran, les options sont les suivantes:

- **Nom de l'application** : c'est le nom de l'application qui sera visible pour l'utilisateur.
- **Nom du package** : il s'agit du nom du package. Il doit être unique, ce qui signifie qu'il ne doit pas utiliser le même nom de package que les autres applications du Google Play Store.
- **Icône de l'application** : C'est l'icône qui sera visible pour l'utilisateur, équivalente à @ drawable / ic_launcher utilisée dans les projets Android Studio ou Eclipse.
- **Numéro de version** : le numéro de version est utilisé par Google Play pour le contrôle de version. Lorsque vous souhaitez publier un fichier APK pour une version mise à jour de votre application, vous devez ajouter 1 à ce numéro pour chaque nouvelle mise à niveau.

- **Nom de la version** : c'est le nom de la version qui sera affiché à l'utilisateur.
- **Emplacement d'installation** : Ceci détermine où votre APK sera installé, dans le stockage de l'appareil ou sur la carte SD.
- **Autorisations requises** : vous déterminez ici les autorisations nécessaires pour votre application.

Options Android

Dans l'écran ci-dessous, vous pouvez configurer les options du compilateur. L'utilisation des bonnes options ici peut réduire considérablement la taille de votre fichier APK et éviter les erreurs.



- **Configuration** : **Active (Release)** .
- **Plate - forme** : **active (n'importe quel processeur)** . Celles-ci sont nécessaires pour construire votre APK pour le Google Play Store. Si la configuration est définie sur Debug, elle ne sera pas acceptée par Google Play.
- **Utiliser le runtime partagé** : **false** . Si vous le définissez sur true, l'APK utilisera Mono Runtime pour s'exécuter. Le Mono Runtime est installé automatiquement lors du débogage via USB, mais pas dans l'APK Release. Si Mono Runtime n'est pas installé sur le périphérique et que cette option est définie sur true dans la version APK, l'application se

bloque.

- **Générez un package (.apk) par ABI sélectionné** : **false** . Créez votre APK pour autant de plates-formes que possible, pour des raisons de compatibilité.
- **Activez Multi-Dex** : **true** , mais vous pouvez le définir sur false si votre application n'est pas très complexe (c'est-à-dire qu'elle contient moins de 65 536 méthodes, voir ici).
- **Activer Proguard** : **true** . Cela active l'outil Proguard qui masque le code Java dans votre application. Notez qu'il ne s'applique pas au code .NET; Si vous voulez masquer le code .NET, vous devez utiliser Dotfuscator . Plus d'informations sur Proguard pour Xamarin.Android peuvent être trouvées ici .
- **Activer l'instrumentation de développeur (débogage et profilage)** : **false** pour Release APK.
- **Liaison** : **SDK et assemblages d'utilisateurs** . Cela rendra le Xamarin Linker pour supprimer toutes les classes inutilisées du SDK et de votre code, réduisant ainsi la taille de l'APK.

Important

Xamarin.Linker peut parfois supprimer des classes qui ne semblent pas être utilisées par votre code, surtout si elles se trouvent dans le Core du projet (bibliothèque PCL). Pour éviter cela, vous pouvez soit définir la liaison avec "Sdk Assemblies Only" ou utiliser l'attribut Preserve dans vos classes, par exemple:

PreserveAttribute.cs

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

Dans une classe:

```
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;
        }
```

```
    }
}
```

- **Architectures prises en charge** : **sélectionnez tout** pour des raisons de compatibilité.

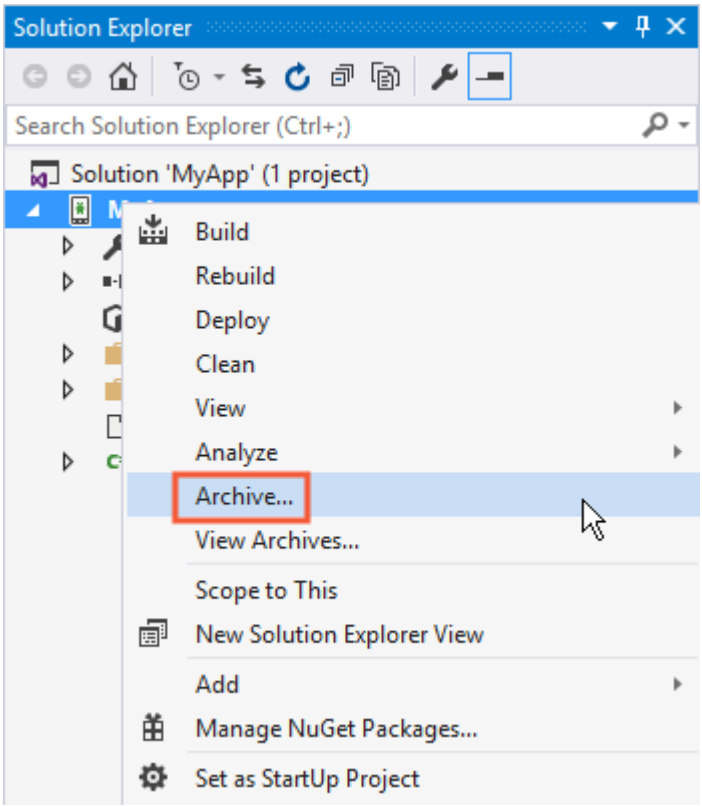
Après avoir tout configuré, reconstruisez le projet pour vous assurer qu'il se construit correctement.

Création du mode APK pour Release

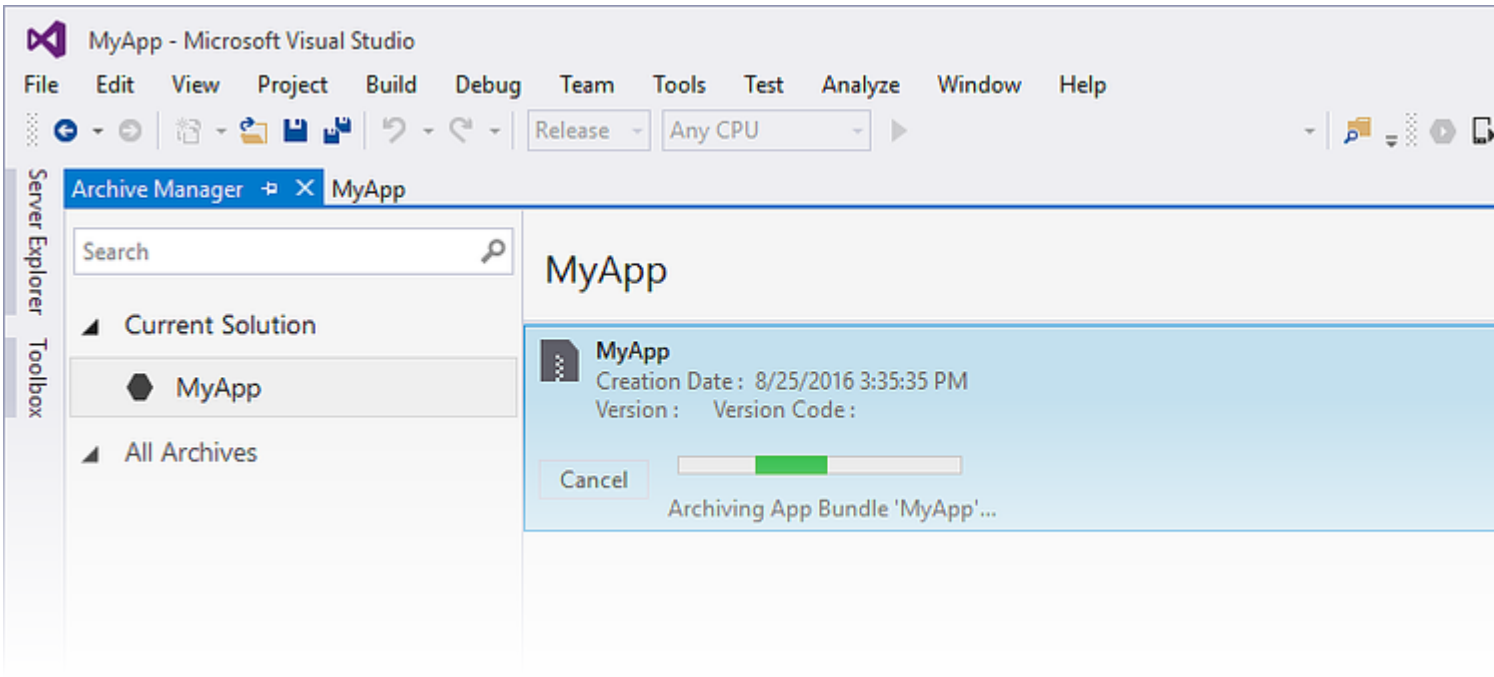
Vous avez fini de configurer votre projet Android pour Release. Le tutoriel ci-dessous montre comment générer l'APK dans Visual Studio. Un tutoriel complet de la documentation Xamarin peut être trouvé ici:

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

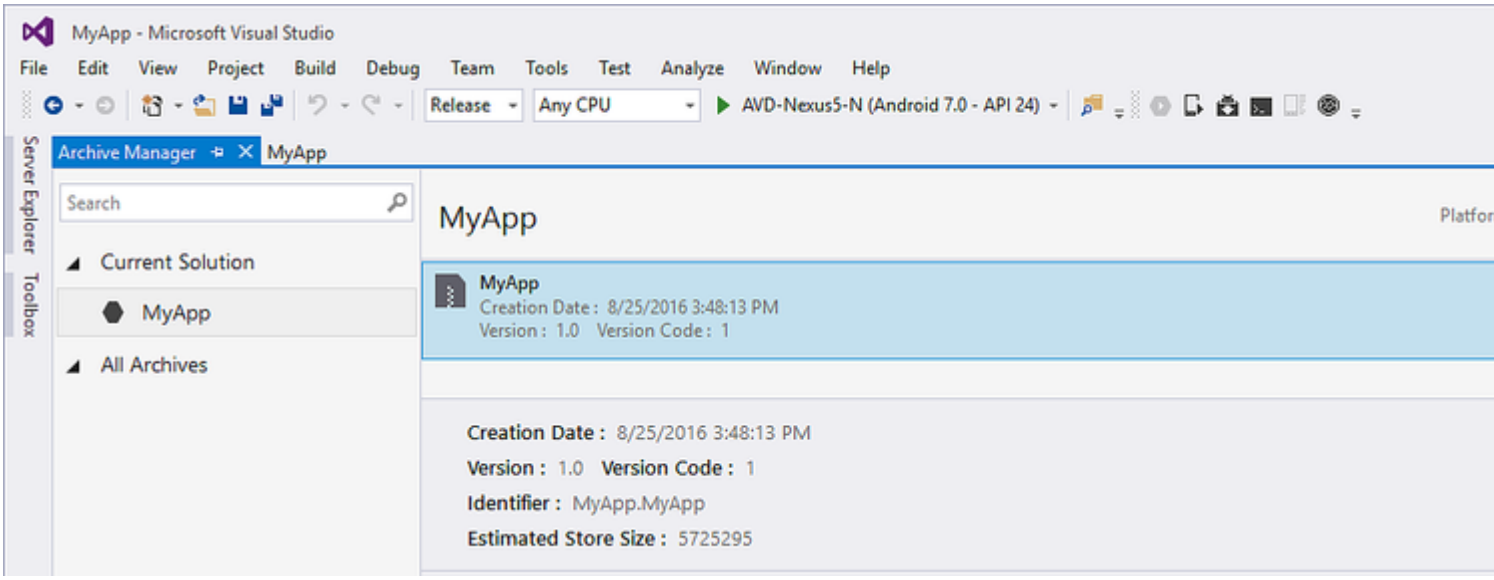
Pour créer le fichier APK, cliquez avec le bouton droit sur le projet Xamarin.Android dans l'Explorateur de solutions et sélectionnez Archiver ...



Cela ouvrira le gestionnaire d'archives et archivera le projet, en préparant le fichier APK.

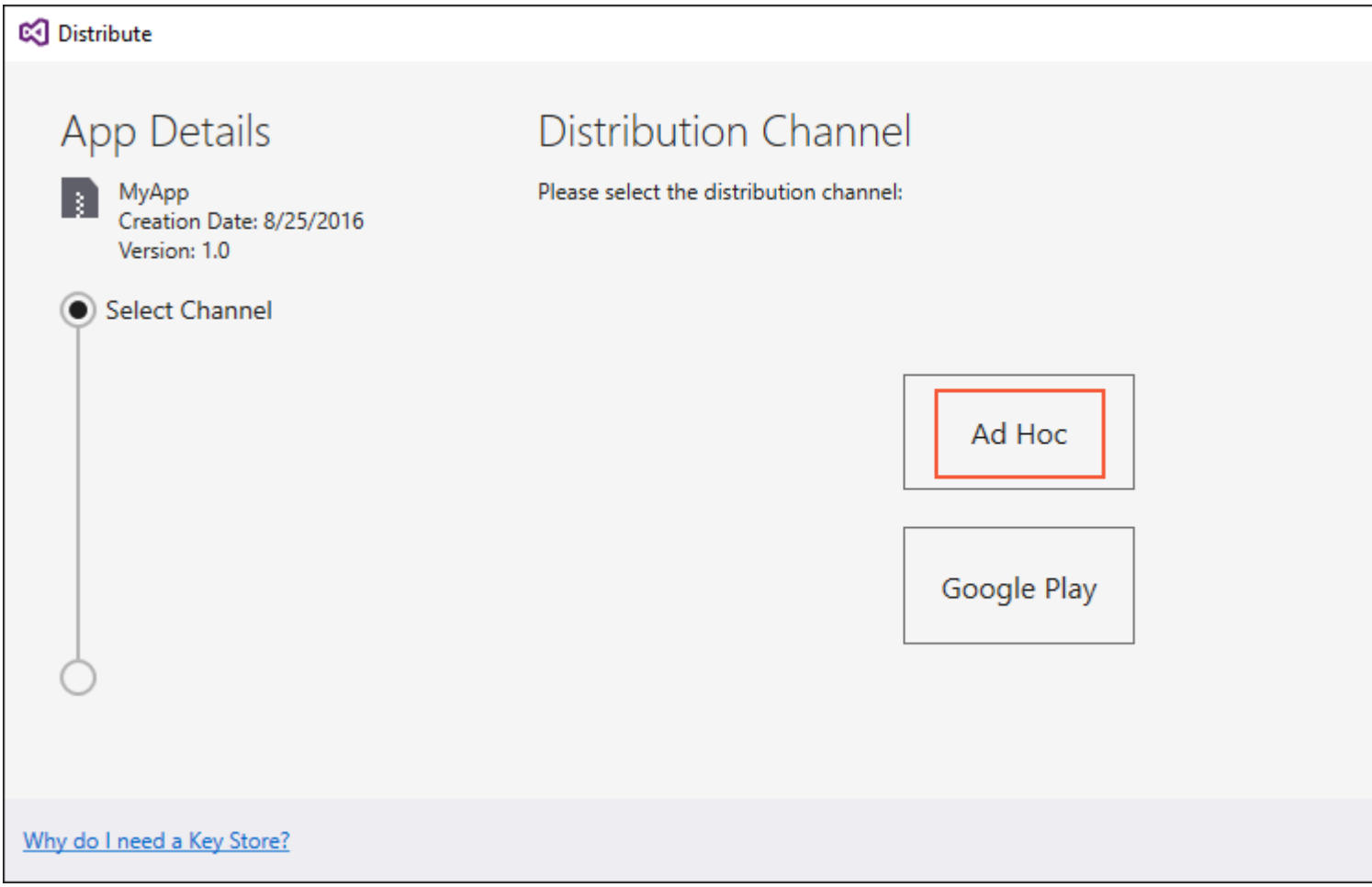


Une fois l'archivage du projet terminé, cliquez sur Distribuer ... pour continuer.

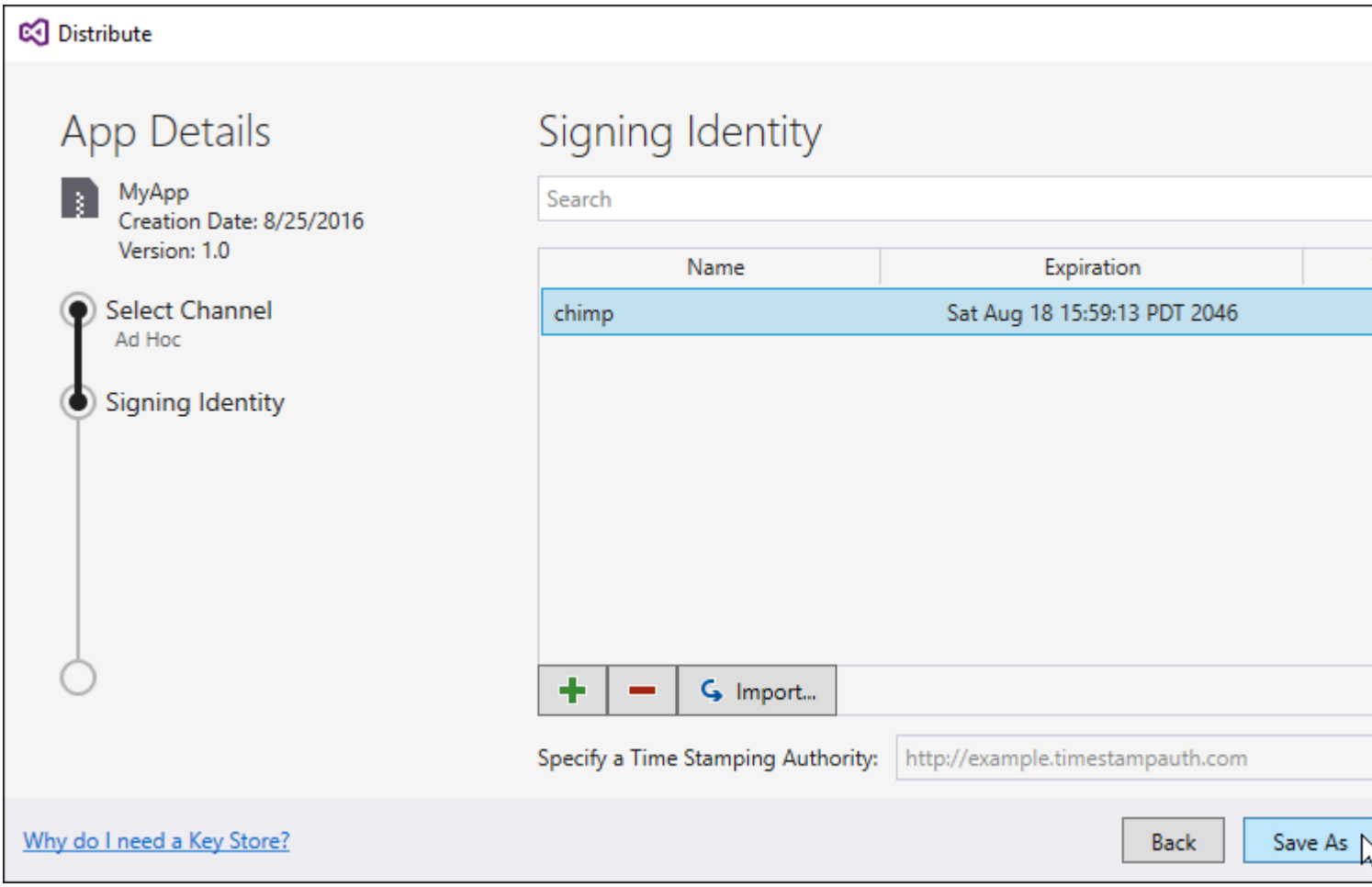


L'écran Distribuer vous présente deux options: Ad-hoc et Google Play. Le premier va créer un APK et l'enregistrer sur votre ordinateur. La seconde publiera directement l'application dans Google Play.

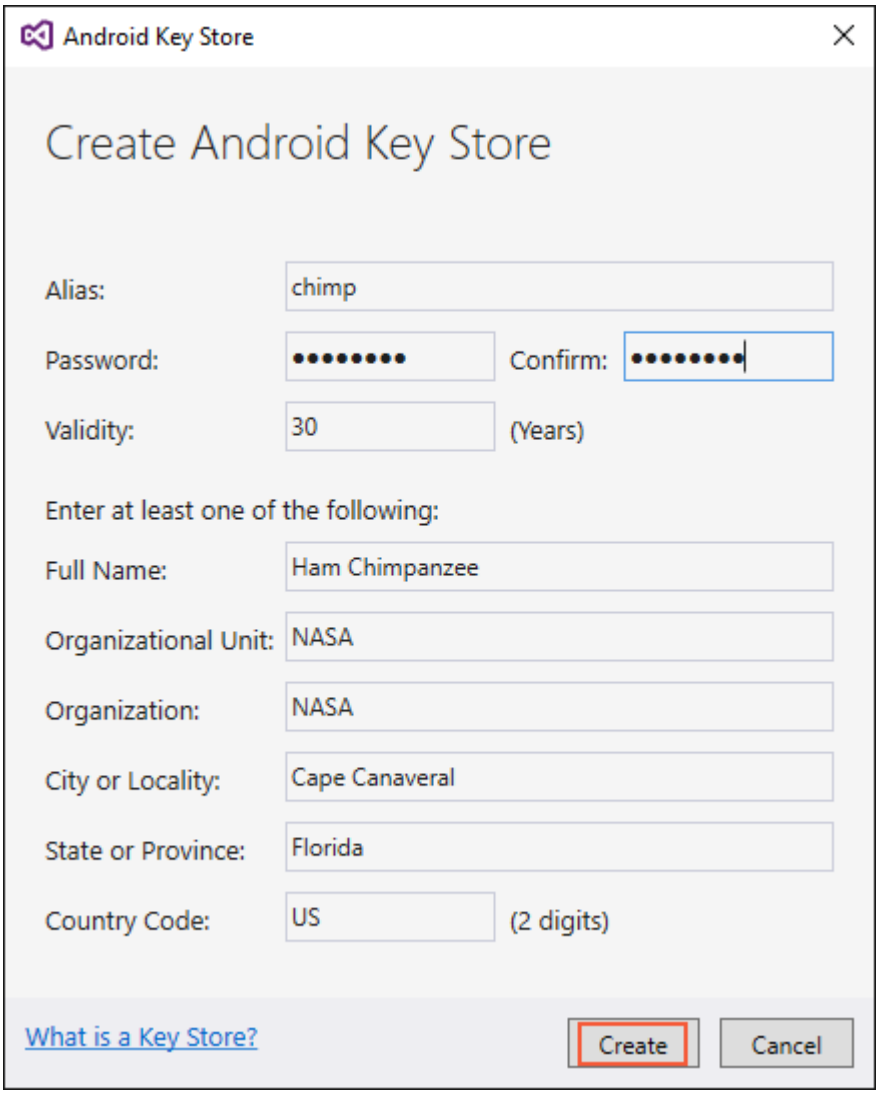
Le choix du premier est recommandé, vous pouvez donc tester l'APK sur d'autres appareils si vous le souhaitez.



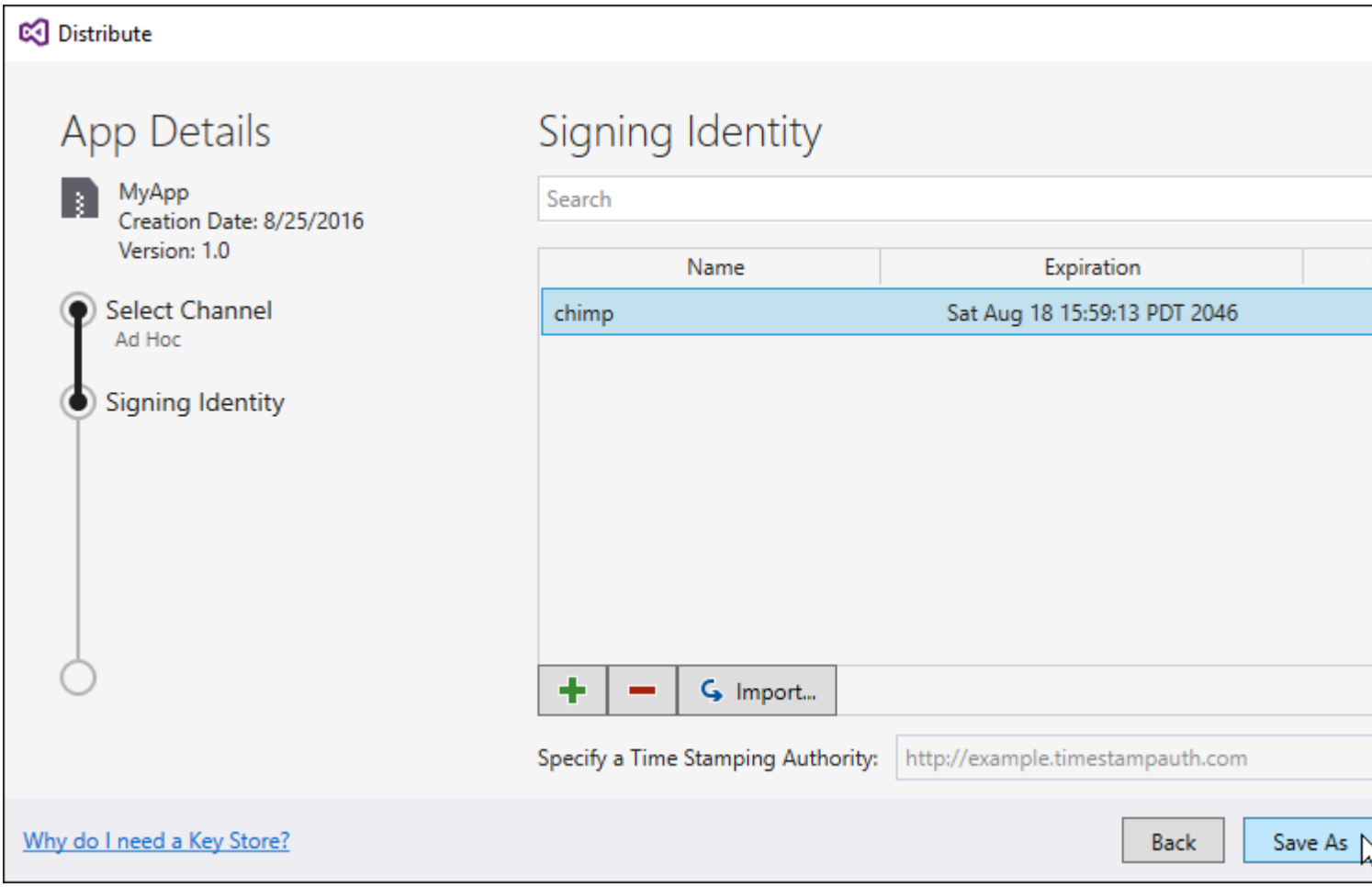
Dans l'écran suivant, un Key Store Android est nécessaire pour signer l'APK. Si vous en avez déjà un, vous pouvez l'utiliser en cliquant sur Importer ...; Si vous ne le faites pas, vous pouvez créer un nouveau magasin de clés Android en cliquant sur +.

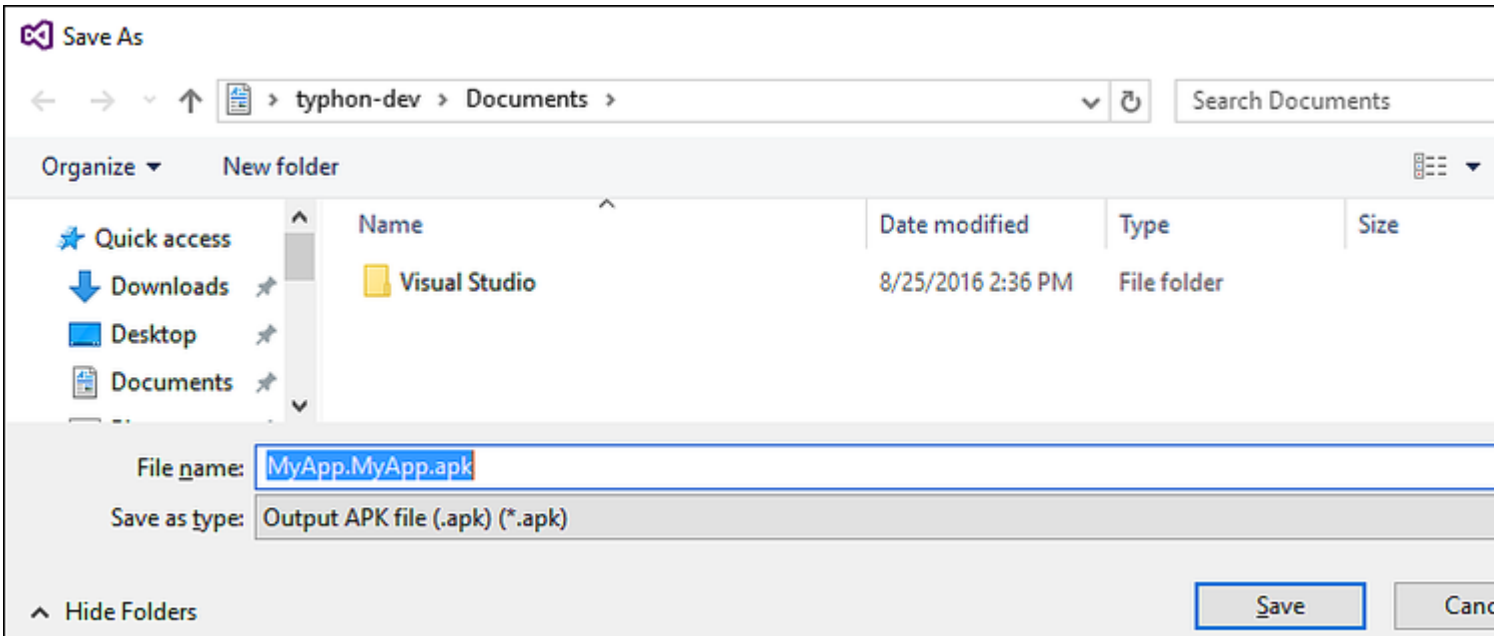


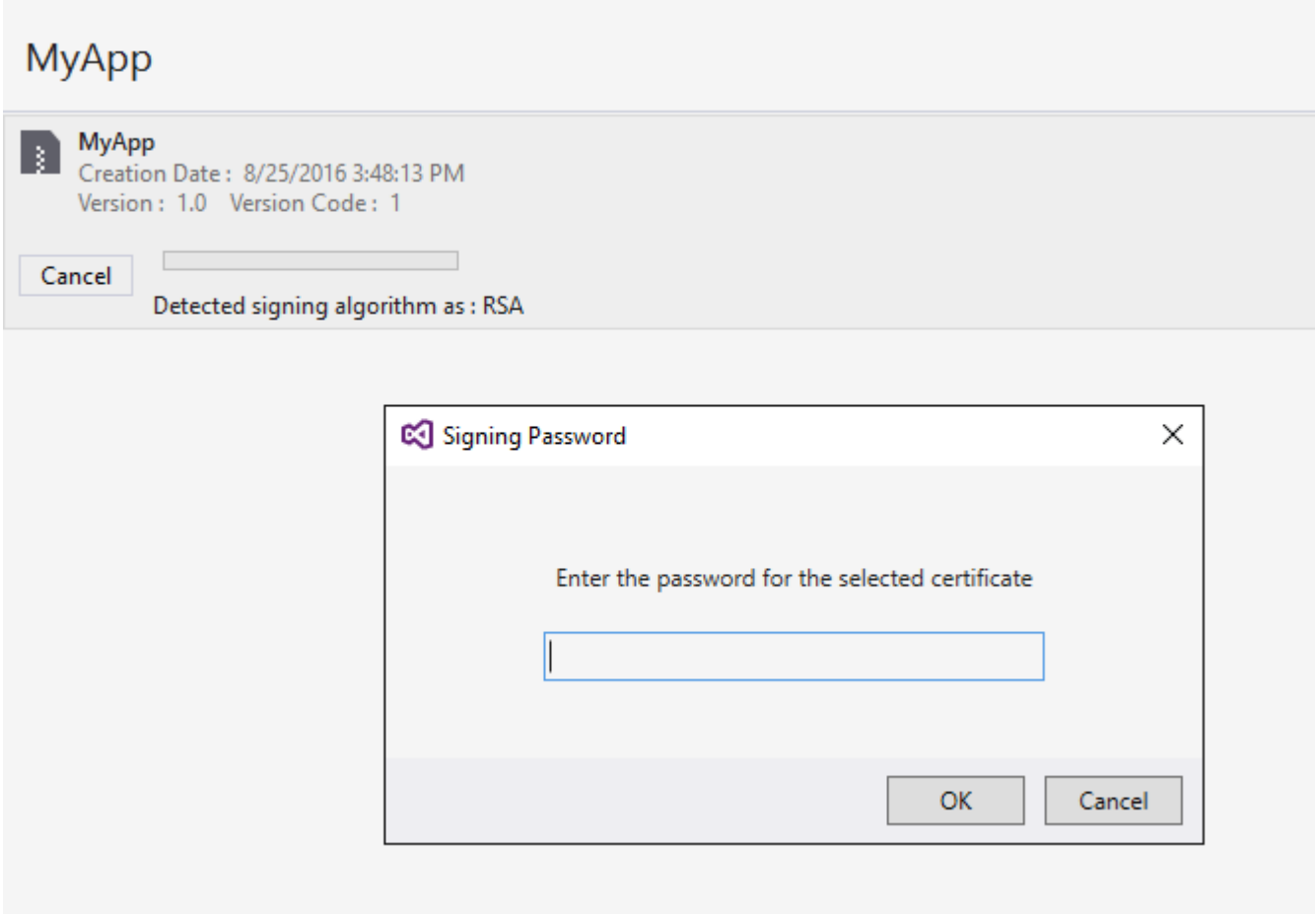
Création d'un nouvel écran Android Key Store:



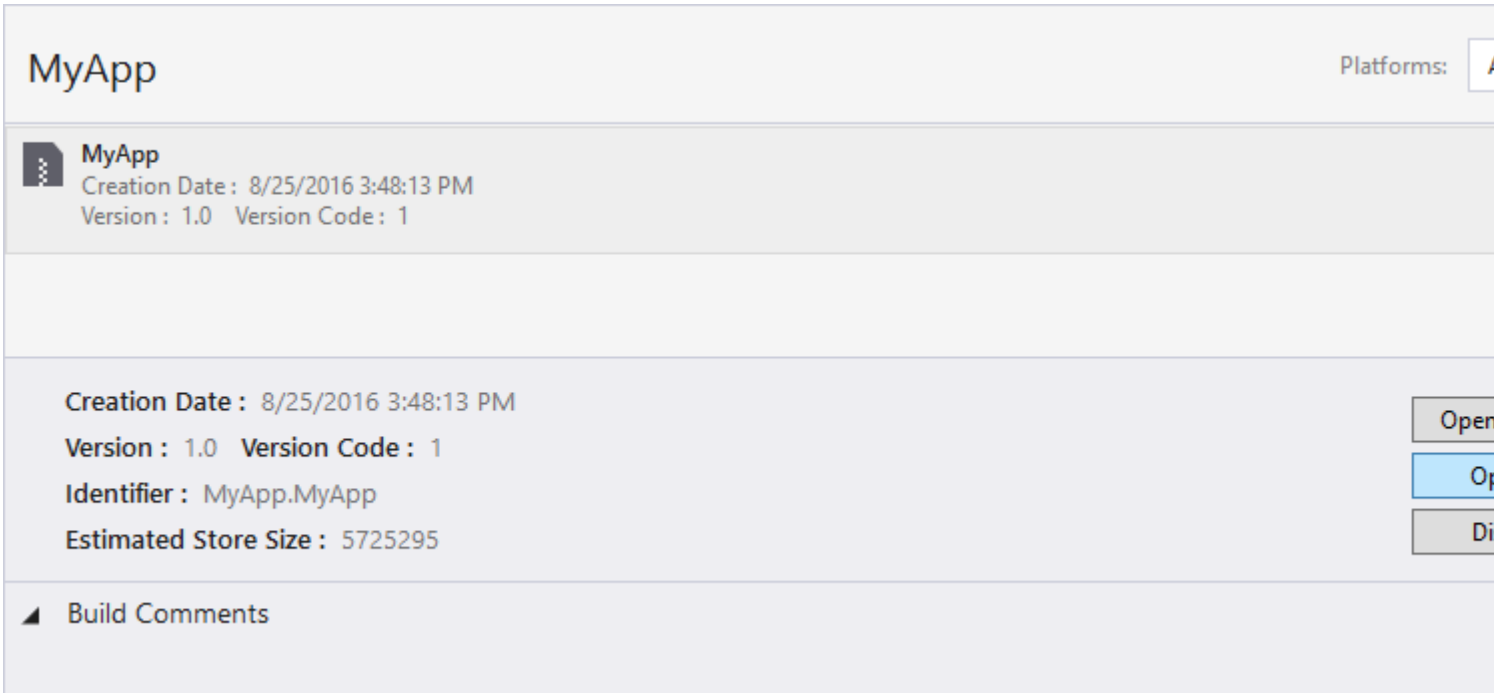
Pour créer l'APK, cliquez sur Enregistrer sous. Vous pouvez être invité à saisir le mot de passe du magasin de clés.

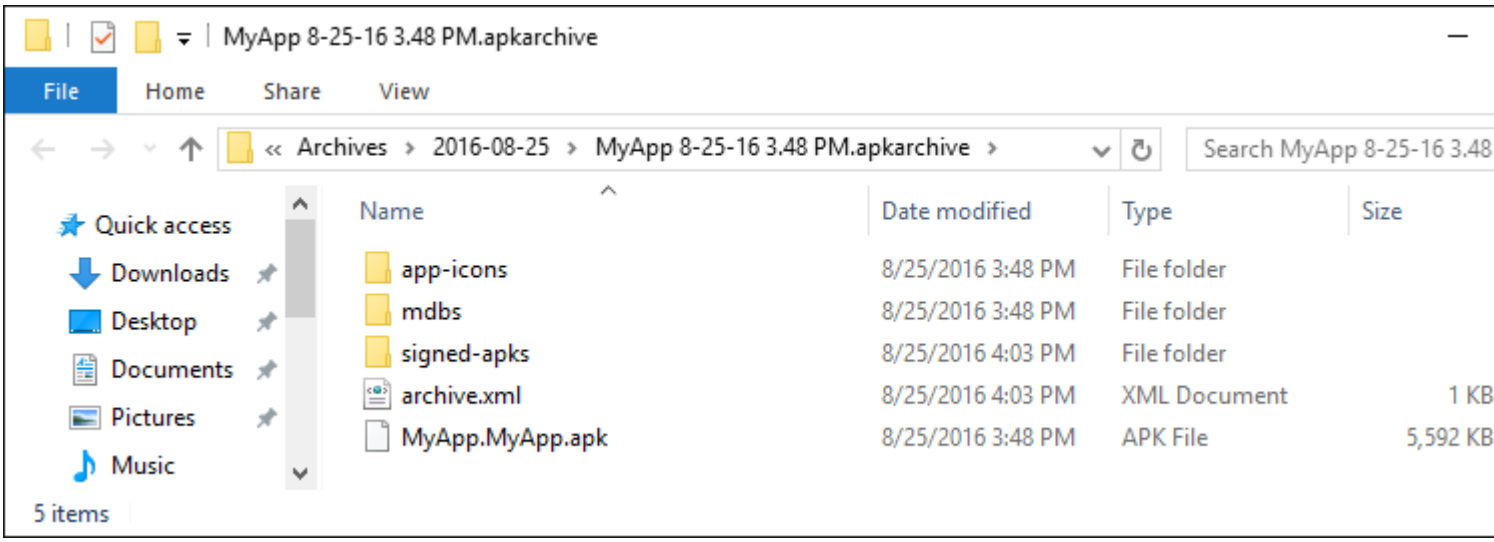

Distribute
App Details
MyApp
Creation Date: 8/25/2016
Version: 1.0
Select Channel
Ad Hoc
Signing Identity
Signing Identity
Search
Name
Expiration
chimp
Sat Aug 18 15:59:13 PDT 2046
Import...
Specify a Time Stamping Authority:
http://example.timestampauth.com
Why do I need a Key Store?
Back
Save As


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



Lorsque cela est terminé, vous pouvez cliquer sur Ouvrir un dossier sur l'écran Archives pour voir le fichier APK généré.





Activation de MultiDex dans votre APK Xamarin.Android

MultiDex est une bibliothèque dans l'APK Android qui permet à l'application d'avoir plus de 65 536 méthodes.

Les APK Android ont des fichiers exécutables Dalvik (.dex) qui contiennent les bytecodes générés à partir de votre code Java. Chaque fichier .dex peut contenir jusqu'à 65 536 méthodes (2 ^ 16).

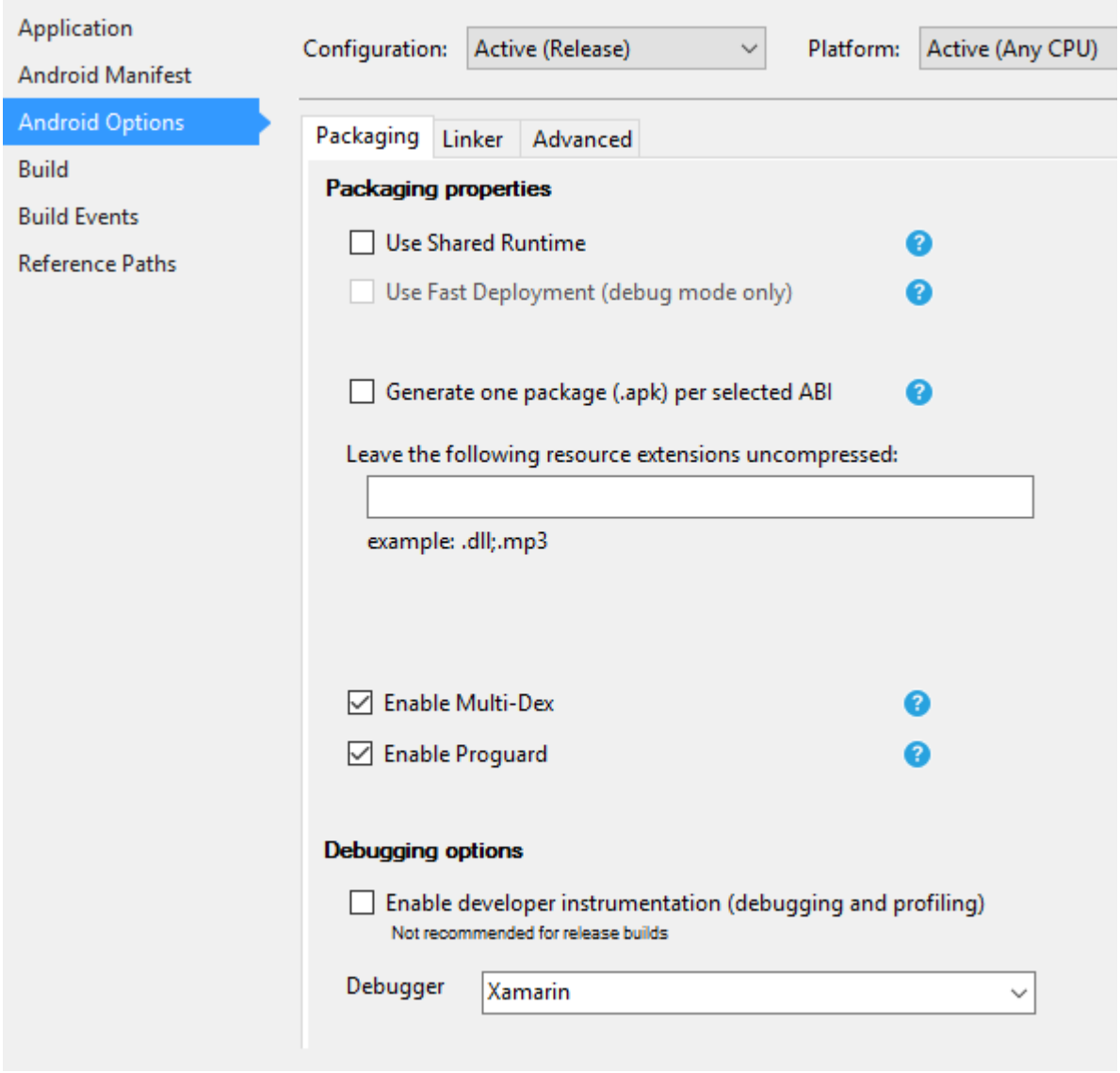
Les versions d'Android OS antérieures à Android 5.0 Lollipop (API 21) utilisent le moteur d'exécution Dalvik, qui ne prend en charge qu'un seul fichier .dex par APK, en limitant à 65 536 méthodes par APK. À partir d'Android 5.0, le système d'exploitation Android utilise ART runtime, qui peut prendre en charge plusieurs fichiers .dex par APK, en évitant la limite.

Pour dépasser la limite des méthodes 65k dans les versions Android inférieures à l'API 21, les développeurs doivent utiliser la bibliothèque de support MultiDex. Le MultiDex crée des fichiers classes.dex supplémentaires (classes2.dex, classes3.dex, ...) en les référençant dans le fichier classes.dex. Lorsque l'application commence à charger, elle utilise une classe MultiDexApplication pour charger les fichiers .dex supplémentaires.

Si votre application Android vise une version minimale du SDK au-dessus ou égale à l'API 21 (Android 5.0 Lollipop), il n'est pas nécessaire d'utiliser la bibliothèque MultiDex, car le système d'exploitation gère nativement les fichiers .dex supplémentaires. Cependant, si, pour des raisons de compatibilité, le développeur souhaite prendre en charge un ancien système d'exploitation Android, il doit alors utiliser la bibliothèque MultiDex.

Comment utiliser MultiDex dans votre application Xamarin.Android

Tout d'abord, pour activer MultiDex dans votre application Xamarin.Android, accédez à votre projet Propriétés -> Options Android -> Conditionnement -> Activer Multi-Dex, comme dans l'écran d'impression ci-dessous:



Ensuite, vous devez créer une classe MultiDexApplication dans votre application. Dans la racine du projet, créez une nouvelle classe (dans l'Explorateur de solutions, cliquez avec le bouton droit sur le projet, Ajouter .. -> Classe ou Maj + Alt + C). Dans le nouveau fichier de classe, copiez le code suivant, en remplaçant l'exemple d'espace de noms par le nom de votre espace de noms de projet 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); }
        }
    }
}
```

Code source ici.

Si vous développez dans Visual Studio pour Windows, il existe également un bogue dans les outils de construction du SDK Android que vous devez corriger pour créer correctement les fichiers classes.dex lors de la construction de votre projet.

Accédez à votre dossier SDK Android, ouvrez le dossier build-tools et il y aura des dossiers avec les numéros des compilateurs du SDK Android, tels que:

C: \ android-sdk \ build-tools \ 23.0.3 \

C: \ android-sdk \ build-tools \ 24.0.1 \

C: \ android-sdk \ build-tools \ 25.0.2 \

À l'intérieur de chacun de ces dossiers, il y a un fichier appelé **mainClassesDex.bat** , un script de commandes utilisé pour créer les fichiers classes.dex. Ouvrez chaque fichier mainClassesDex.bat avec un éditeur de texte (Notepad ou Notepad ++) et dans son script, recherchez et remplacez le bloc:

```
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
```

Avec le bloc:

```
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
```

Source ici

Enregistrez chaque mainClassesDex.bat dans l'éditeur de texte après les modifications.

Après les étapes ci-dessus, vous devriez pouvoir créer votre application Xamarin.Android avec MultiDex.

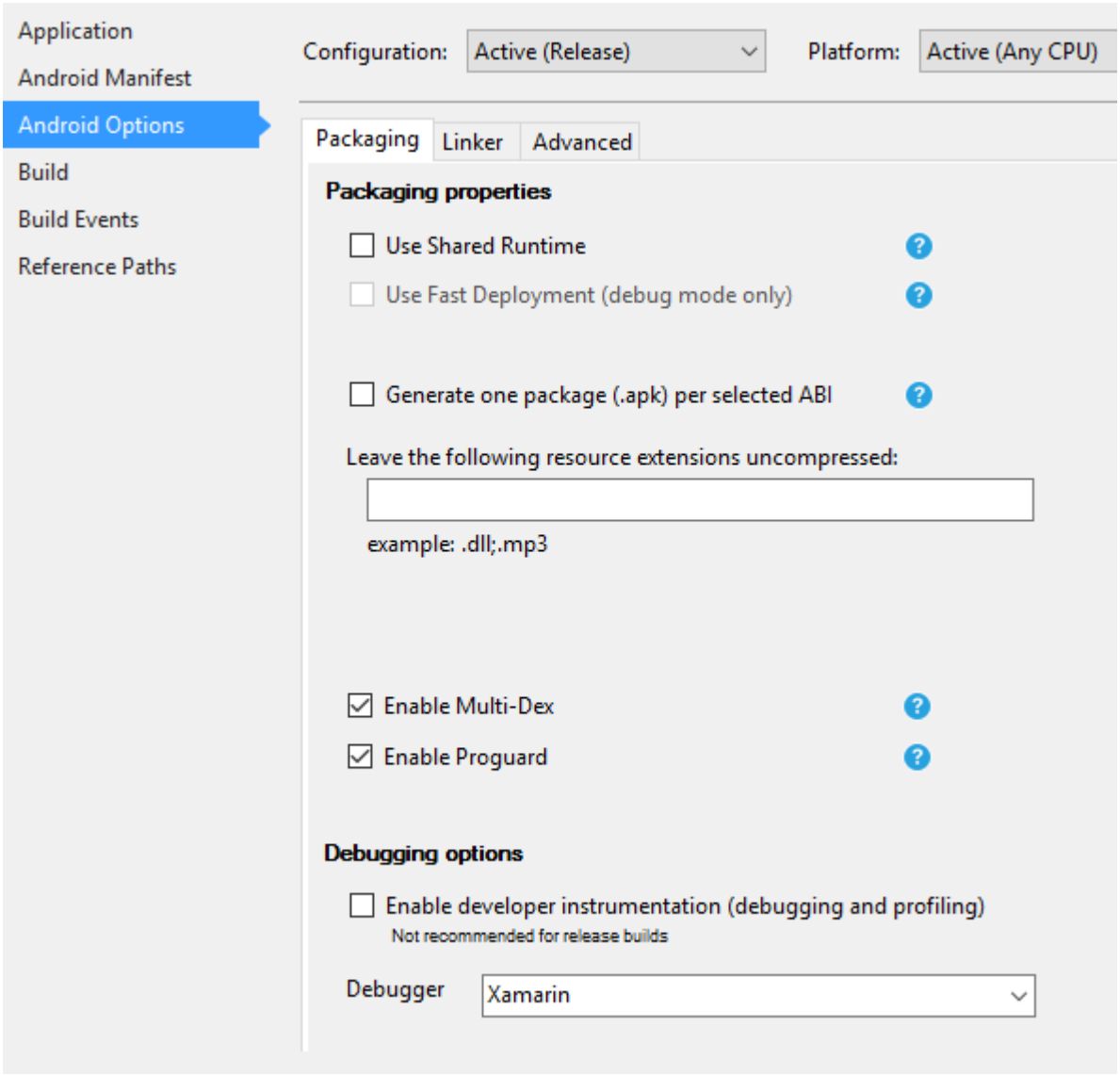
Activation de ProGuard dans votre APK Xamarin.Android

ProGuard est un outil utilisé dans le processus de création pour optimiser et masquer le code Java de votre APK, ainsi que pour supprimer les classes inutilisées. Le résultat APK lors de l'utilisation de ProGuard aura une taille plus petite et sera plus difficile à désosser (décompilation).

ProGuard peut également être utilisé dans les applications Xamarin.Android et réduira également la taille du fichier APK et masquera le code Java. Sachez cependant que l'obfuscation de ProGuard s'applique uniquement au code Java. Pour brouiller le code .NET, le développeur doit utiliser Dotfuscator ou des outils similaires.

Comment utiliser ProGuard dans votre application Xamarin.Android

Tout d'abord, pour activer ProGuard dans votre application Xamarin.Android, accédez à votre projet Propriétés -> Options Android -> Conditionnement -> Activer ProGuard, comme dans l'écran d'impression ci-dessous:

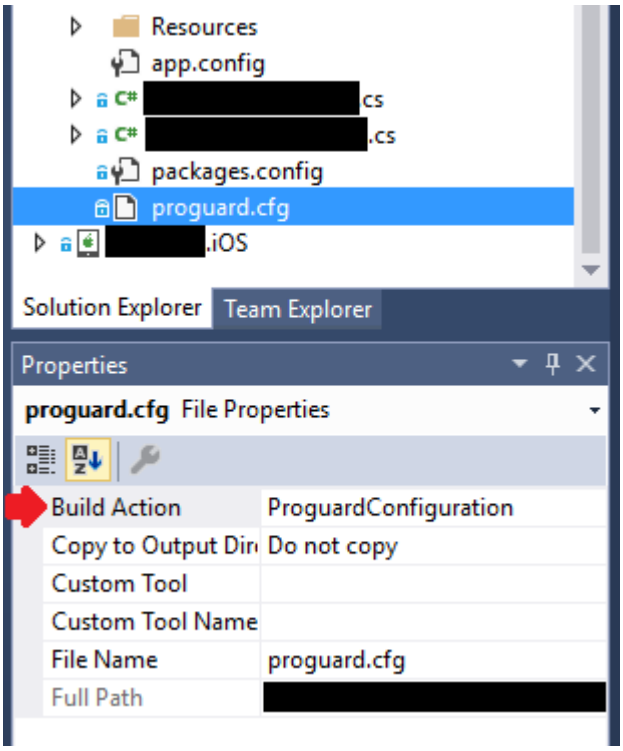


Cela permet à ProGuard de créer votre application.

Xamarin.Android, par défaut, définit ses propres configurations pour ProGuard, qui se trouvent dans les dossiers `obj/Debug/proguard` ou `obj/Release/proguard` , dans les fichiers `proguard_project_primary.cfg` , `proguard_project_references.cfg` et `proguard_xamarin.cfg` . Les trois fichiers sont combinés en tant que configurations pour ProGuard et ils sont automatiquement créés par Xamarin lors de la création.

Si le développeur souhaite personnaliser davantage les options de ProGuard, il peut créer un fichier dans la racine du projet nommé `proguard.cfg` (les autres noms sont également valides, tant que l'extension est .cfg) et définir son action de génération sur ProguardConfiguration, comme

dans l'image ci-dessous:



Dans le fichier, des options ProGuard personnalisées peuvent être insérées, telles que `-dontwarn` , `-keep class` et autres .

Important

À ce jour (avril 2017), le SDK Android généralement téléchargé possède une ancienne version de ProGuard, ce qui peut entraîner des erreurs lors de la création de l'application à l'aide de Java 1.8. Lors de la construction, la liste des erreurs affiche le message suivant:

```
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]
```

Source ici

Pour résoudre ce problème, vous devez télécharger la version la plus récente de ProGuard (ici) et copier le contenu du fichier .zip dans `android-sdk\tools\proguard\` . Cela mettra à jour ProGuard et le processus de construction devrait fonctionner sans problème.

Après cela, vous devriez être capable de créer votre application Xamarin.Android avec ProGuard.

Bugs "mystérieux" liés à ProGuard et Linker

Vous avez créé une application géniale et l'avez testée dans Debug, avec de bons résultats. Tout fonctionnait bien!

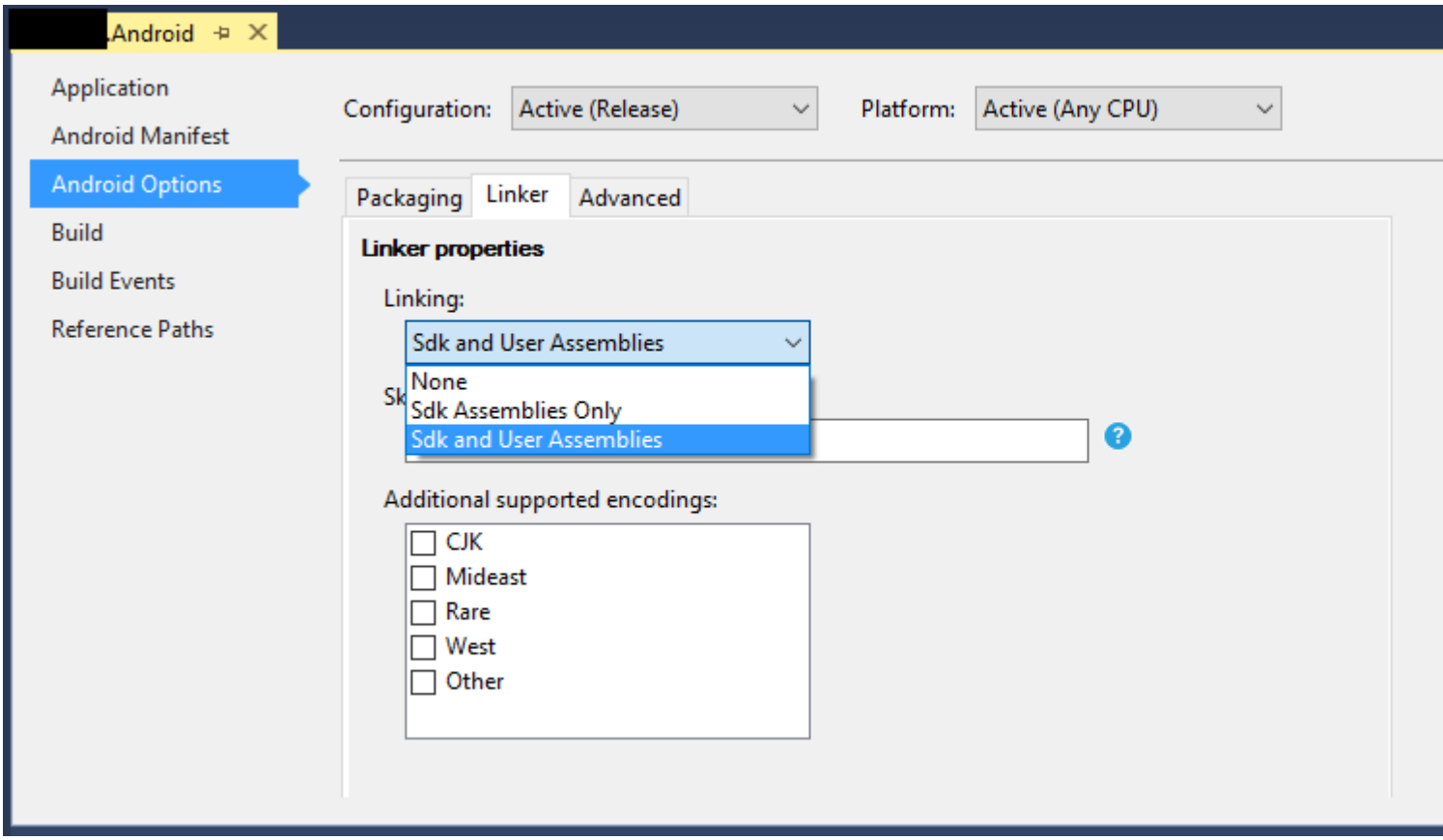
Mais alors, vous avez décidé de préparer votre application pour la publication. Vous avez

configuré MultiDex, ProGuard et Linker, puis il a cessé de fonctionner.

Ce tutoriel a pour but de vous aider à identifier les problèmes courants liés à ProGuard et Linker qui peuvent causer des bogues mystérieux.

Comprendre Xamarin.Linker

Xamarin.Linker est un outil du processus de construction qui supprime le code et les classes inutilisés **de votre code .NET (pas le code Java)** . Dans les propriétés de votre projet -> Options Android -> Linker, il y aura une boîte de sélection Liaison avec les options:



Aucun : aucun code n'est supprimé.

Assemblages SDK uniquement : cette option permet au Xamarin.Linker de rechercher le code inutilisé uniquement dans les bibliothèques Xamarin. **Cette option est sûre.**

Sdk et User Assemblies : cette option permet au Xamarin.Linker de rechercher le code inutilisé dans les bibliothèques Xamarin et dans le code du projet (y compris les PCL, les composants Xamarin et les packages NuGet). **Cette option n'est pas toujours sûre!**

Lors de l'utilisation de l'option Sdk et User Assemblies, Xamarin.Linker peut penser que des parties du code ne sont pas utilisées lorsque, en réalité, elles sont très utilisées! Cela peut empêcher certaines bibliothèques de fonctionner correctement et causer des bogues dans votre application.

Pour que le Xamarin.Linker ne supprime pas le code, il existe 3 options:

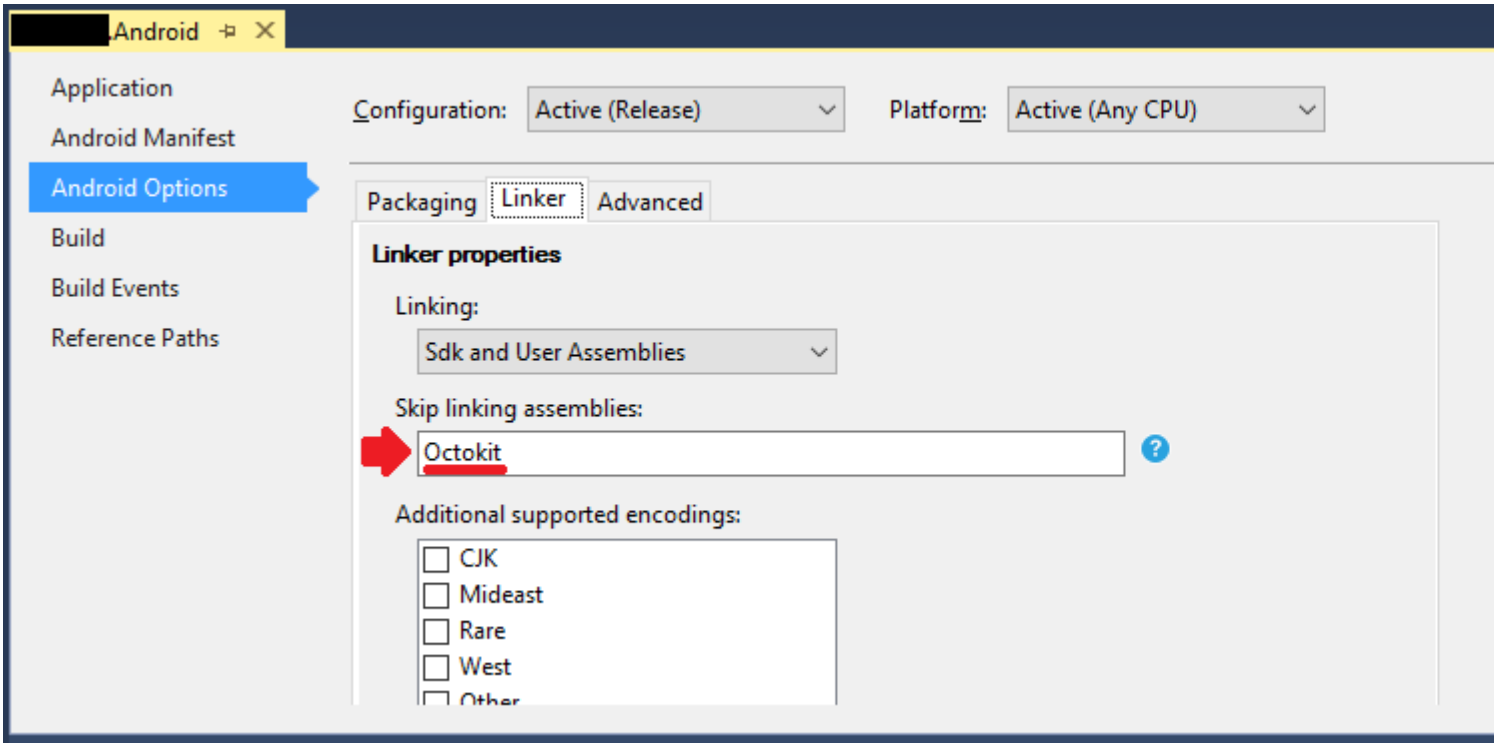
1. Définition de l'option de liaison sur None ou Sdk Assemblies uniquement;
2. Ignorer les assemblages de liens;
3. Utiliser l'attribut Preserve.

Exemple pour 2. Ignorer les assemblages de liaison:

Dans l'exemple ci-dessous, l'utilisation de Xamarin.Linker a provoqué un arrêt du fonctionnement de NuGet Package (Octokit), car il ne pouvait plus se connecter à Internet:

```
[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 ---
```

Pour que la bibliothèque recommence à fonctionner, il était nécessaire d'ajouter le nom de référence du package dans le champ Ignorer les assemblages de liaison, situé dans le projet -> Propriétés -> Options Android -> Linker, comme dans l'image ci-dessous:



Après cela, la bibliothèque a commencé à fonctionner sans aucun problème.

Exemple pour 3. Utilisation de l'attribut Preserve:

Xamarin.Linker perçoit comme un code inutilisé principalement le code des classes de modèle dans le noyau de votre projet.

Pour que la classe soit préservée pendant le processus de liaison, vous pouvez utiliser l'attribut Preserve.

Tout d'abord, créez dans le noyau de votre projet une classe nommée **PreserveAttribute.cs** , insérez le code suivant et remplacez l'espace de noms par l'espace de noms de votre projet:

PreserveAttribute.cs:

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

Dans chaque classe de modèle du noyau de votre projet, insérez l'attribut Préserver comme dans l'exemple ci-dessous:

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;
        }
    }
}
```

Après cela, le processus de liaison ne supprimera plus le code conservé.

Comprendre ProGuard

ProGuard est un outil du processus de construction qui supprime le code et les classes inutilisés **de votre code Java** . Il obscurcit et optimise également le code.

Cependant, ProGuard peut parfois supprimer le code qu'il considère inutilisé, alors que ce n'est pas le cas. Pour éviter cela, le développeur doit déboguer l'application (dans le moniteur de périphérique Android et dans le débogage Visual Studio) et détecter quelle classe a été supprimée, puis configurer le fichier de configuration ProGuard pour conserver la classe.

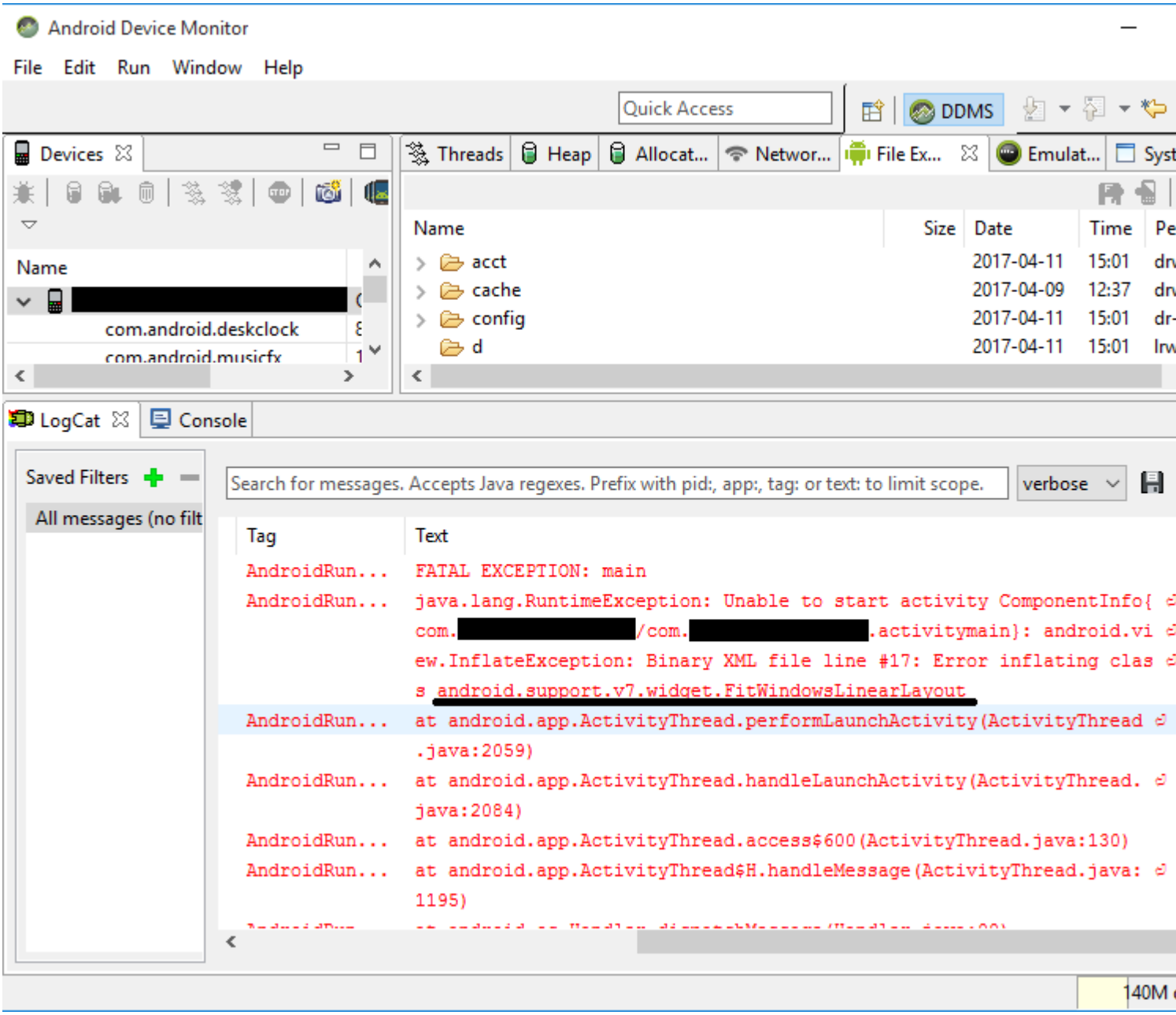
Exemple

Dans l'exemple ci-dessous, ProGuard a supprimé deux classes (Android.Support.V7.Widget.FitWindowsLinearLayout et Android.Support.Design.Widget.AppBarLayout) utilisées dans les fichiers de mise en page AXML, mais perçues comme non utilisées dans le code. La suppression a provoqué une exception ClassNotFoundException dans le code Java lors du rendu de la disposition de l'activité:

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 indiquant une erreur lors de la création de la mise en page dans SetContentView:



Pour corriger cette erreur, il était nécessaire d'ajouter les lignes suivantes au fichier de configuration ProGuard du projet:

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

Après cela, plus aucune erreur n'a été affichée lors de la création de la mise en page.

Avertissements ProGuard

ProGuard affiche parfois des avertissements dans la liste des erreurs après la création de votre projet. Bien qu'ils posent la question de savoir si votre application est correcte ou non, tous leurs avertissements n'indiquent pas de problèmes, en particulier si votre application est correctement créée.

Par exemple, lorsque vous utilisez la bibliothèque Picasso : lorsque vous utilisez ProGuard, cela

peut afficher des avertissements tels que `okio.Okio: can't find referenced class (...)` ou d' `can't write resource [META-INF/MANIFEST.MF] (Duplicate zip entry [okhttp.jar:META-INF/MANIFEST.MF]) (...)` , mais l'application est construite et la bibliothèque fonctionne sans problème.

Lire Publier votre APK Xamarin.Android en ligne: https://riptutorial.com/fr/xamarin-android/topic/9601/publier-votre-apk-xamarin-android

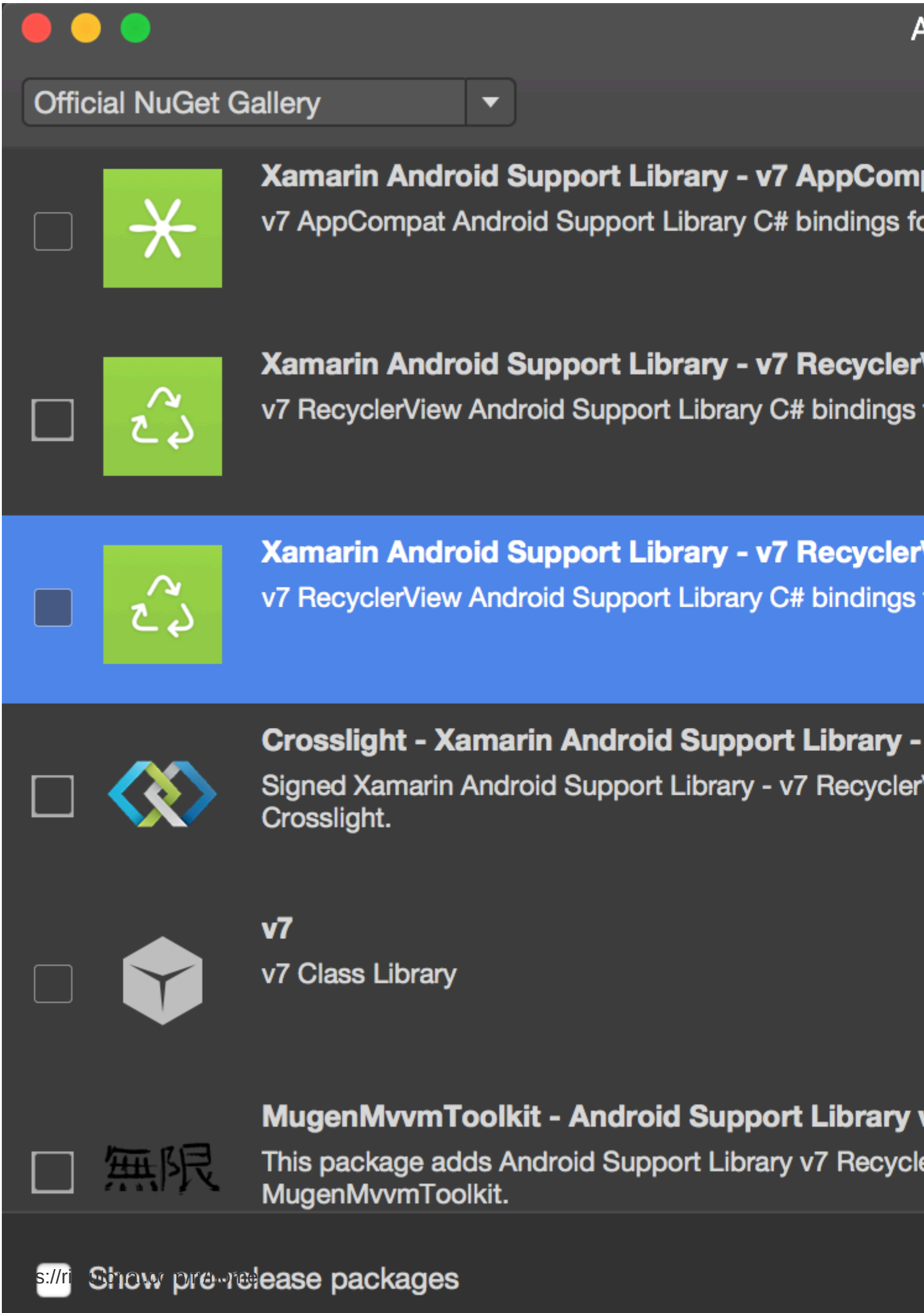
Chapitre 10: RecyclerView

Examples

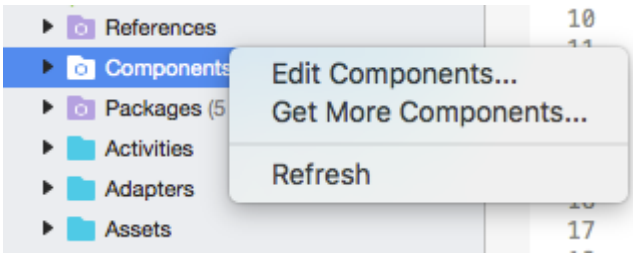
RecyclerView Basics

Ceci est un exemple d'utilisation de la `Android Support Library V7 RecyclerView` . Les bibliothèques de support sont généralement recommandées car elles fournissent des versions rétrocompatibles des nouvelles fonctionnalités, fournissent des éléments d'interface utilisateur utiles qui ne sont pas inclus dans la structure et fournissent une gamme d'utilitaires pouvant être utilisés par les applications.

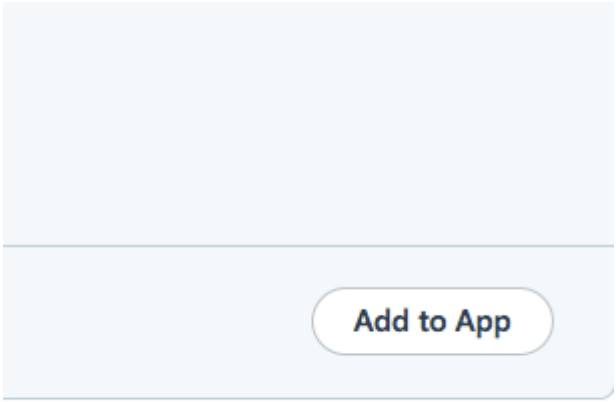
Pour obtenir le `RecyclerView` , nous allons installer les paquets Nuget nécessaires. Tout d'abord, nous rechercherons `v7 recyclerview` . Faites défiler la liste jusqu'à ce que nous `Xamarin Android Support Library – v7 RecyclerView` . Sélectionnez-le et cliquez sur **Ajouter un package** .


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

est disponible en tant que composant Xamarin. Pour ajouter le composant, cliquez avec le bouton droit sur `Components` du projet Android dans l'explorateur de solutions et cliquez sur `Get More Components` .



Dans la fenêtre Component Store qui apparaît, recherchez RecyclerView. Dans la liste de recherche, sélectionnez `Android Support Library V7 RecyclerView` . Cliquez ensuite sur `Add to App` . Le composant est ajouté au projet.



La prochaine étape consiste à ajouter RecyclerView à une page. Dans le `axml` (layout), nous pouvons ajouter `RecyclerView` comme ci-dessous.

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

RecyclerView nécessite la configuration d'au moins deux classes d'assistance pour l'implémentation standard de base, à savoir: `Adapter` et `ViewHolder` . `Adapter` gonfle les dispositions d'éléments et lie les données aux vues affichées dans RecyclerView. ViewHolder recherche et stocke les références de vue. Le porte-vues permet également de détecter les clics sur les vues d'objets.

Voici un exemple de base de la classe d'adaptateur

```
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;
        }
    }
}
```

Dans la méthode `OnCreateViewHolder` , nous `OnCreateViewHolder` abord une vue et créons une instance de la classe ViewHolder. Cette instance doit être renvoyée. Cette méthode est appelée par l'adaptateur lorsqu'il nécessite une nouvelle instance de ViewHolder. Cette méthode ne sera pas invoquée pour chaque cellule. Une fois que RecyclerView a suffisamment de cellules pour remplir la vue, il réutilisera les anciennes cellules qui défilent hors de la vue pour d'autres cellules.

Le rappel `OnBindViewHolder` est `OnBindViewHolder` par l'adaptateur pour afficher les données à la position spécifiée. Cette méthode doit mettre à jour le contenu de itemView pour refléter l'élément à la position donnée.

Comme la cellule ne contient qu'un seul `TextView` , nous pouvons avoir un ViewHolder simple comme ci-dessous.

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


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

La prochaine étape consiste à connecter des objets dans `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;
}
```

La classe LayoutManager est chargée de mesurer et de positionner les vues d'éléments dans RecyclerView et de déterminer la stratégie de recyclage des vues d'éléments qui ne sont plus visibles pour l'utilisateur. Avant `RecyclerView` , nous devions utiliser `ListView` pour organiser les cellules comme dans une liste déroulante verticale et `GridView` pour afficher les éléments dans une grille à deux dimensions et défilant. Mais maintenant, nous pouvons réaliser les deux avec RecyclerView en définissant un LayoutManger différent. `LinearLayoutManager` organise les cellules comme dans ListView et `GridLayoutManager` organise les cellules en mode Grid.

RecyclerView avec les événements Click

Cet exemple montre comment définir Click EventHandlers dans un Xamarin.Android RecyclerView.

Dans Android Java , la façon de configurer un écouteur pour un clic consiste à utiliser un onClickListener pour la vue sur laquelle l'utilisateur cliquera, comme ceci:

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

Toutefois, dans Xamarin.Android, la manière de configurer un écouteur pour un événement

Click consiste à **ajouter** un gestionnaire d'événements, de l'une des manières suivantes:

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
}
```

Notez que **le gestionnaire d'événements est ajouté, pas défini.** Si le gestionnaire d'événements Click Click est ajouté à l'intérieur d'une méthode GetView à partir d'un adaptateur GridView / ListView ou d'une méthode OnBindViewHolder à partir de RecyclerView.Adapter, un nouveau EventHandler sera ajouté chaque fois que la vue d'élément est créée. Après avoir fait défiler plusieurs fois, plusieurs EventHandlers seront ajoutés et lorsque la vue sera cliquée, ils seront tous déclenchés.

Pour éviter ce problème, les EventHandlers doivent être désabonnés et abonnés ultérieurement dans les méthodes GetView ou OnBindViewHolder. De plus, ils doivent utiliser le numéro **3.** pour définir le gestionnaire d'événements, sinon il ne sera pas possible de se désinscrire des gestionnaires d'événements.

Un exemple de RecyclerView.Adapter avec des événements Click est illustré ci-dessous:

```
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);
    }

}
```

Lire RecyclerView en ligne: https://riptutorial.com/fr/xamarin-android/topic/3452/recyclerview

Chapitre 11: Toasts

Examples

Message Toast de base

Tout d'abord, instanciez un objet Toast avec l'une des méthodes `MakeText()` . Cette méthode prend trois paramètres: le `Context` l'application, le message texte et la durée du toast. Il retourne un objet Toast correctement initialisé. Vous pouvez afficher la notification de toast avec `Show()` , comme illustré dans l'exemple suivant:

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

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

Cet exemple montre tout ce dont vous avez besoin pour la plupart des notifications de toast. Vous devriez rarement avoir besoin d'autre chose. Vous pouvez, cependant, vouloir placer le pain grillé différemment ou même utiliser votre propre mise en page au lieu d'un simple message texte. Les sections suivantes décrivent comment faire ces choses.

Vous pouvez également enchaîner vos méthodes, appeler comme une seule ligne et éviter de tenir l'objet Toast, comme ceci:

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

Pour plus d'informations, reportez-vous à la documentation Android plus complète sur le sujet.

Messages de pain grillé coloré

Parfois , nous voulons donner des informations supplémentaires à nos utilisateurs avec des couleurs (par exemple rouge signifie quelque chose de mal est arrivé) Nous pouvons changer la couleur de fond du message de pain grillé à l' aide la fixation d' un filtre de couleur à la vue que notre pain nous donner (ici j'utilise un 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();
```

Et aussi, nous pouvons changer la couleur du texte si l'arrière-plan est clair ou sombre:

```
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
```

Changer la position du pain grillé

Nous pouvons changer notre pain grillé en utilisant la méthode SetGravity. Cette méthode prend trois paramètres: d'abord la gravité du pain grillé à l'écran et deux autres définissent le décalage du pain grillé par rapport à la position de départ (définie par le premier paramètre):

```
//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();
```

Lire Toasts en ligne: https://riptutorial.com/fr/xamarin-android/topic/3550/toasts

Chapitre 12: Xamarin.Android - Comment créer une barre d'outils

Remarques

Chère équipe,

Je pense que c'est bon de mentionner la documentation officielle d'Android où le contrôle de la barre d'outils est expliqué en détail:

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

Il y a aussi du contenu intéressé sur la bibliothèque Android.Support.v7 utilisée dans l'exemple:

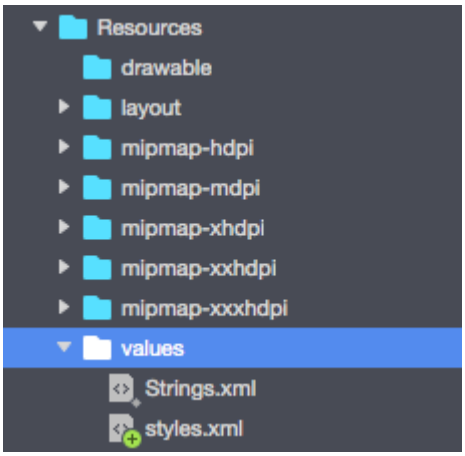
https://developer.android.com/training/appbar/index.html

Examples

Ajouter une barre d'outils à l'application Xamarin.Android

Tout d'abord, vous devez ajouter la bibliothèque Xamarin.Android.Support.V7.AppCompat pour NuGet: https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat/

Dans le dossier "values" sous "Resources", ajoutez un nouveau fichier xml appelé "styles.xml":

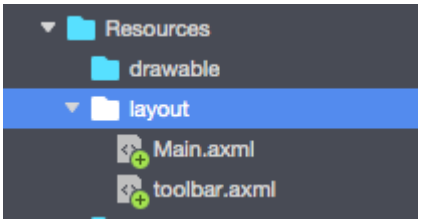


Le fichier "styles.xml" doit contenir le code ci-dessous:

```
<?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. -->
```

La prochaine étape consiste à ajouter le fichier "toolbar.axml" contenant la définition du contrôle de la barre d'outils au dossier "layout":



Ajouter le code ci-dessous pour définir la barre d'outils:

```
<?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" />
```

Maintenant, ouvrez le fichier "Main.axml" et ajoutez le code ci-dessous juste en dessous de la balise de fermeture pour la première mise en page. Votre code devrait ressembler à celui-ci:

```
<?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>
```

Maintenant, vous devez ajouter des informations sur le thème que votre application utilise. Ouvrez le fichier "AndroidManifest" et ajoutez les informations de thème à la balise "application":

```
<application android:theme="@style/MyTheme" android:allowBackup="true"
```

```
android:icon="@mipmap/icon" android:label="@string/app_name">
```

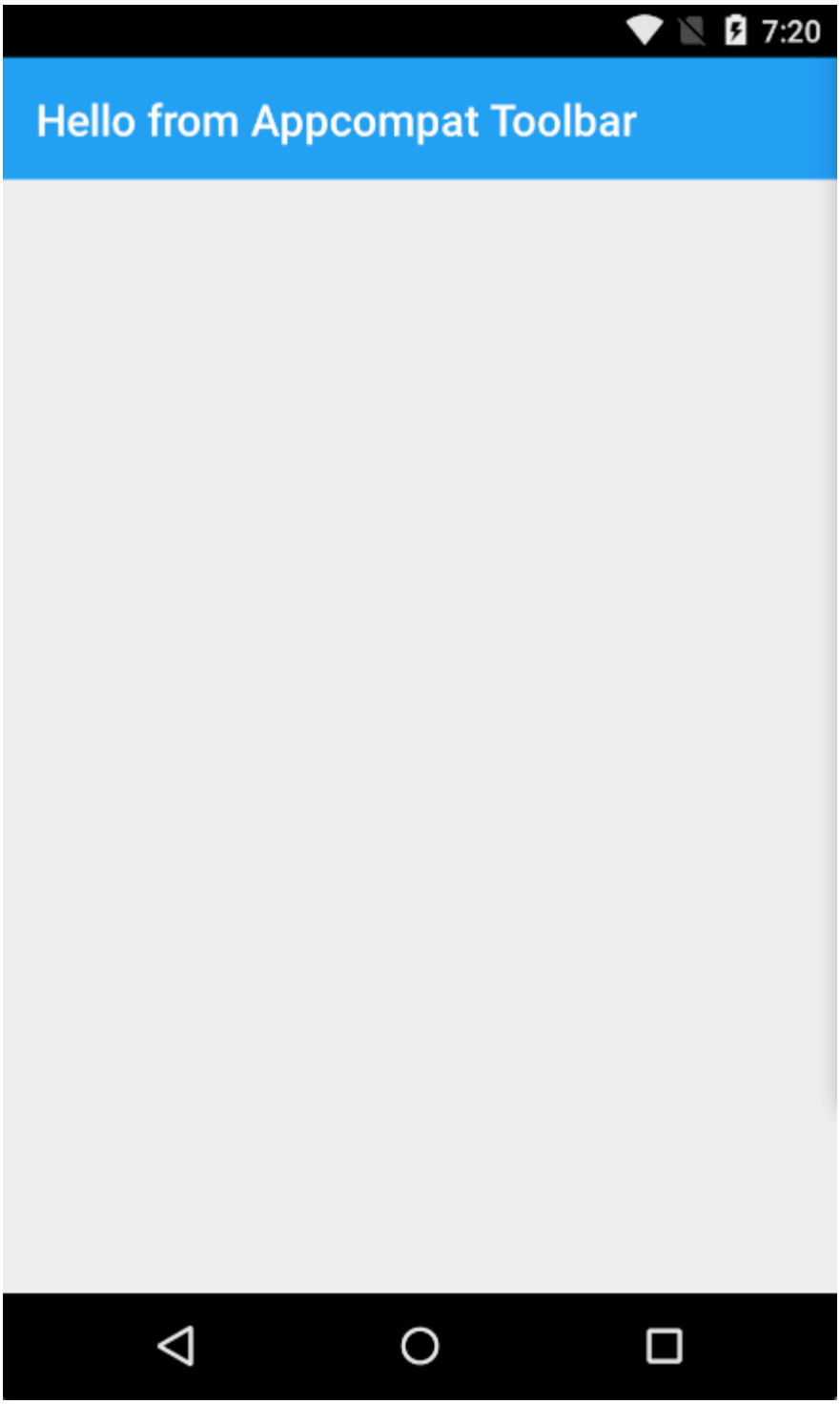
La dernière étape consiste à connecter la barre d'outils dans le fichier d'activité. Ouvrez le fichier "MainActivity.cs". Vous devez changer la dérivation de "Activité" à "AppCompatActivity". Faites maintenant référence à la barre d'outils et définissez-la comme barre d'outils par défaut pour l'activité dans la méthode "OnCreate". Vous pouvez également définir le titre:

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

Toute la méthode devrait ressembler à la suivante:

```
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";
    }
```

Reconstruire le projet et le lancer pour voir le résultat:



Lire Xamarin.Android - Comment créer une barre d'outils en ligne: https://riptutorial.com/fr/xamarin-android/topic/4755/xamarin-android---comment-creer-une-barre-d-outils

Chapitre 13: Xamarin.Android - Communication Bluetooth

Introduction

Dans **Xamarin.Android,** les propriétés **BluetoothSocket.InputStream** et **BluetoothSocket.OutputStream** sont automatiquement converties dans **System.IO.Stream** . Dans le cas du protocole de communication interactif, lorsque le serveur ne répond que lorsque le client lui parle, System.IO.Stream n'est pas bon car il ne possède aucune méthode ou propriété pour obtenir le nombre d'octets de réponse disponibles avant de lire la réponse.

Paramètres

Paramètre	Détails
prise	Une instance d'objet BluetoothSocket. Socket doit être ouvert avant d'appeler cette méthode.
cmd	Commande en tant que tableau d'octets à envoyer au périphérique BT.
_mx	Comme cette méthode utilise une ressource matérielle, il est préférable de l'appeler à partir d'un thread de travail distinct. Ce paramètre est une instance de l'objet System.Threading.Mutex et permet de synchroniser le thread avec d'autres threads appelant éventuellement cette méthode.
temps libre	Temps d'attente en millisecondes entre les opérations d'écriture et de lecture.

Examples

Envoyer et recevoir des données depuis et vers un périphérique Bluetooth via socket

L'exemple ci-dessous utilise les types Android.Runtime.InputStreamInvoker et Android.Runtime.OutputStreamInvoker pour obtenir Java.IO.InputStream et Java.IO.OutputStream . Une fois que nous avons une instance **Java.IO.InputStream** , nous pouvons utiliser sa méthode **.Available ()** pour obtenir le nombre d'octets de réponse disponibles que nous pouvons utiliser dans la méthode **.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;
}
```

Lire Xamarin.Android - Communication Bluetooth en ligne: https://riptutorial.com/fr/xamarin-android/topic/10844/xamarin-android---communication-bluetooth

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Xamarin.Android	Amy Burns, Community, Jon Douglas, Kevin Montrose, Ryan Weaver
2	Balayage de codes à barres à l'aide de la bibliothèque ZXing dans les applications Xamarin	GvSharma
3	Comment corriger l'orientation d'une image capturée depuis un appareil Android	Daniel Krzyczkowski
4	Cycle de vie de l'application - Xamarin.Andorid	CDrosos, Daniel Krzyczkowski, Steven Mark Ford
5	Dialogues	JimBobBennett, Pilatus
6	Fixations	EJoshuaS, Jon Douglas, jonp, Matthew, Prashant C, Sven-Michael Stübe
7	Liste personnalisée	user3814750
8	Publier votre APK Xamarin.Android	Alexandre
9	RecyclerView	Alexandre, Matthew, Ryan Alford, Sreeraj, Zverev Eugene
10	Toasts	GONeale, Matthew, Piet, user2912553
11	Xamarin.Android - Comment créer une barre d'outils	Daniel Krzyczkowski, tylerjgarland
12	Xamarin.Android - Communication Bluetooth	Ladislav