Splash Screen

03/01/2018 · 4 minutes to read · Contributors

In this article

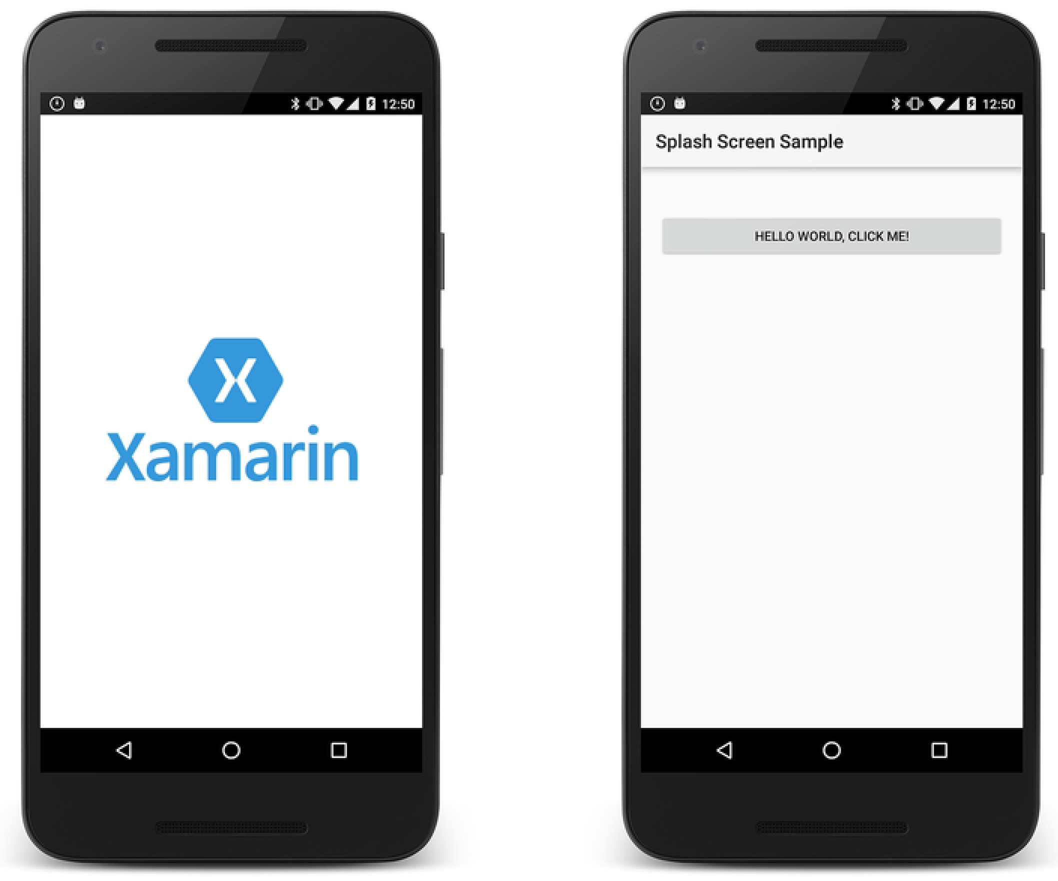
An Android app takes some time to to start up, especially when the app is first launched on a device. A splash screen may display start up progress to the user or to indicate branding.

Overview

An Android app takes some time to to start up, especially during the first time the app is run on a device (sometimes this is referred to as a *cold start*). The splash screen may display start up progress to the user, or it may display branding information to identify and promote the application.

This guide discusses one technique to implement a splash screen in an Android application. It covers the following steps:

1. Creating a drawable resource for the splash screen.
2. Defining a new theme that will display the drawable resource.
3. Adding a new Activity to the application that will be used as the splash screen defined by the theme created in the previous step.



Requirements

This guide assumes that the application targets Android API level 15 (Android 4.0.3) or higher. The application must also have the **Xamarin.Android.Support.v4** and **Xamarin.Android.Support.v7.AppCompat** NuGet packages added to the project.

All of the code and XML in this guide may be found in the SplashScreen sample project for this guide.

Implementing A Splash Screen

The quickest way to render and display the splash screen is to create a custom theme and apply it to an Activity that exhibits the splash screen. When the Activity is rendered, it loads the theme and applies the drawable resource (referenced by the theme) to the background of the activity. This approach avoids the need for creating a layout file.

The splash screen is implemented as an Activity that displays the branded drawable, performs any initializations, and starts up any tasks. Once the app has bootstrapped, the splash screen Activity starts the main Activity and removes itself from the application back stack.

Creating a Drawable for the Splash Screen

The splash screen will display an XML drawable in the background of the splash screen Activity. It is necessary to use a bitmapped image (such as a PNG or JPG) for the image to display.

In this guide, we use a Layer List to center the splash screen image in the application. The following snippet is an example of a `drawable` resource using a `layer-list`:

XML

Copy

```xml
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <color android:color="@color/splash_background"/>
  </item>
  <item>
    <bitmap
        android:src="@drawable/splash"
        android:tileMode="disabled"
        android:gravity="center"/>
  </item>
</layer-list>
```

This `layer-list` will center the splash screen image **splash.png** on a background specified by the `@color/splash_background` resource. Place this file in the **Resources/drawable** folder (for example, **Resources/drawable/splash_screen.xml**).

After the splash screen drawable has been created, the next step is to create a theme for the splash screen.

Implementing a Theme

To create a custom theme for the splash screen Activity, edit (or add) the file **values/styles.xml** and create a new `style` element for the splash screen. A sample **values/style.xml** file is shown below with a `style` named **MyTheme.Splash**:

XML

Copy

```xml
<resources>
  <style name="MyTheme.Base" parent="Theme.AppCompat.Light">
  </style>

  <style name="MyTheme" parent="MyTheme.Base">
  </style>

  <style name="MyTheme.Splash" parent ="Theme.AppCompat.Light.NoActionBar">
    <item name="android:windowBackground">@drawable/splash_screen</item>
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowFullscreen">true</item>
  </style>
</resources>
```

MyTheme.Splash is very spartan – it declares the window background, explicitly removes the title bar from the window, and declares that it is full-screen. If you want to create a splash screen that emulates the UI of your app before the activity inflates the first layout, you can use `windowContentOverlay` rather than `windowBackground` in your style definition. In this case, you must also modify the **splash_screen.xml** drawable so that it displays an emulation of your UI.

Create a Splash Activity

Now we need a new Activity for Android to launch that has our splash image and performs any startup tasks. The following code is an example of a complete splash screen implementation:

C#

Copy

```
[Activity(Theme = "@style/MyTheme.Splash", MainLauncher = true, NoHistory = true)]
public class SplashActivity : AppCompatActivity
{
    static readonly string TAG = "X:" + typeof(SplashActivity).Name;

    public override void OnCreate(Bundle savedInstanceState, PersistableBundle persistentState)
    {
        base.OnCreate(savedInstanceState, persistentState);
        Log.Debug(TAG, "SplashActivity.OnCreate");
    }

    // Launches the startup task
    protected override void OnResume()
    {
        base.OnResume();
        Task startupWork = new Task(() => { SimulateStartup(); });
        startupWork.Start();
    }

    // Simulates background work that happens behind the splash screen
    async void SimulateStartup ()
    {
        Log.Debug(TAG, "Performing some startup work that takes a bit of time.");
        await Task.Delay (8000); // Simulate a bit of startup work.
        Log.Debug(TAG, "Startup work is finished - starting MainActivity.");
        StartActivity(new Intent(Application.Context, typeof (MainActivity)));
    }
}
```

`SplashActivity` explicitly uses the theme that was created in the previous section, overriding the default theme of the application. There is no need to load a layout in `OnCreate` as the theme declares a drawable as the background.

It is important to set the `NoHistory=true` attribute so that the Activity is removed from the back stack. To prevent the back button from canceling the startup process, you can also override `OnBackPressed` and have it do nothing:

C#

Copy

```
public override void OnBackPressed() { }
```

The startup work is performed asynchronously in `OnResume`. This is necessary so that the startup work does not slow down or delay the appearance of the launch screen. When the work has completed, `SplashActivity` will launch `MainActivity` and the user may begin interacting with the app.

This new `SplashActivity` is set as the launcher activity for the application by setting the `MainLauncher` attribute to `true`. Because `SplashActivity` is now the launcher activity, you must edit `MainActivity.cs`, and remove the `MainLauncher` attribute from `MainActivity`:

C#

```
[Activity(Label = "@string/ApplicationName")]
public class MainActivity : AppCompatActivity
{
    // Code omitted for brevity
}
```

Summary

This guide discussed one way to implement a splash screen in a Xamarin.Android application; namely, applying a custom theme to the launch activity.

Related Links

- SplashScreen (sample)
- layer-list Drawable
- Material Design Patterns - Launch Screens