Pamela
Sep 11, 2024 · 8 min read

How to Add a Splash Screen to a Compose Multiplatform App

Updated: Oct 24, 2024

Compose Multiplatform is a technology by JetBrains based on Jetpack Compose, bringing the power of declarative UIs to Android, iOS, desktop, and web. Splash screens give quick branding opportunities at the start of your application. However, splash screens are a native feature as splash screens are displayed even before the first Compose Multiplatform screen is displayed, so these screens need to be done separately and in a platform-specific way for each supported platform.

This article will take you through what to do to add static and animated splash screens to Android and iOS applications. It assumes basic knowledge of creating Kotlin Multiplatform apps for these platforms. The tutorial assumes that you started with a Compose Multiplatform project for Android and iOS created recently with the web wizard at kmp.jetbrains.com, but you can also apply the steps to your own projects, hopefully with minimal trouble.

Let's get started by adding a static splash screen to an Android app.

Repository with code: https://github.com/pahill/cmp-splash-screens

Add a static splash screen to the Android app

Add the splash screen compatibility library

In Android Studio, navigate to the composeApp/src/build.gradle.kts file. Add the splash screen library dependency to the androidMain source set. It should look something like this:

```
    sourceSets {

        androidMain.dependencies {
```

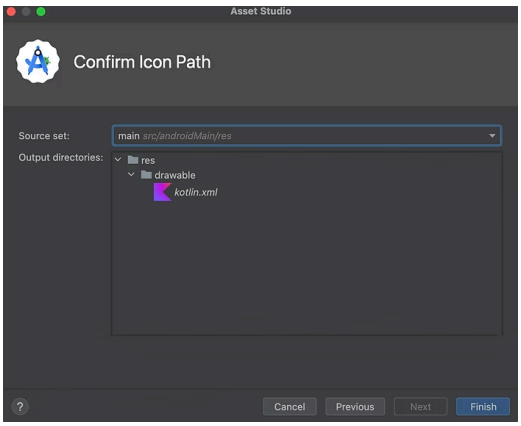
```
            …

            implementation("androidx.core:core-splashscreen:1.0.1")

        }

        commonMain.dependencies {

            …

        }

    }
```

Remember to sync Gradle.

Import the .svg file as a vector asset

In Android Studio, start by importing an .svg file into your project.

1. File > New > Vector Asset.
2. In the Configure Vector Asset dialog, select the Local File radio button.
3. In the Path file selector control, navigate to the .svg file and click on the Open button.
4. Click on the Next button.
5. In the Confirm Icon Path dialog, ensure that the source set and output directories are set correctly as below, and click on the Finish button.
6. In the Project navigator window, ensure the corresponding XML file appears in the output directory as above.



Make the vector drawable fit

The Android splash screen API has some dimension requirements for the splash screen icon:

- Icon with an icon background: must be 240×240 dp and fit within a circle 160 dp in diameter.
- Icon without an icon background: must be 288×288 dp and fit within a circle 192 dp in diameter.

This is because the splash screen API applies a circle mask to the screen. Conforming to the above requirements ensures that the entire logo fits into the circle mask's visible area.

An easy way to do this is as follows:

1. Open the vector drawable XML file, and note the viewport width (android:viewportWidth) and height (android:viewportHeight)
2. Create a top-level group element with the following attributes:
   android:pivotX="x" where x is the value is half android:viewportWidth determined in step 1
   android:pivotY="y" where y is the value is half android:viewportHeight determined in step 2
   android:scaleX="0.4"
   this value is based on the width of your vector asset, mine was 100dp
   android:scaleY="0.4"
   this value is based on the height of your vector asset, mine was 100dp

Create a splash screen theme

It's now necessary to create a customized splash screen theme:

1. Right-click on composeApp/src/androidMain/res/values
2. Select New > Values resource file in the menu
3. Call the new values resource file something like "splash_screen_theme.xml" and click on the OK button
4. Edit the resulting file to look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="Theme.App.Splash" parent="Theme.SplashScreen">
        <item name="windowSplashScreenBackground">
            #202020
        </item>
        <item name="windowSplashScreenAnimatedIcon">
            @drawable/kotlin
        </item>
        <item name="postSplashScreenTheme">
            @android:style/Theme.Material.NoActionBar
        </item>
    </style>
</resources>
```

Let's analyze our new splash screen theme. The splash screen theme, Theme.App.Splash, inherits from the Theme.SplashScreen theme defined by the Android splash screen library. We are overriding three of its attributes:

- windowSplashScreenAnimatedIcon to define the vector drawable acting as our logo
- windowSplashScreenBackground to define the background color of the rest of the screen
- postSplashScreenTheme to define the next theme - this will most probably be the theme of the rest of the application

Apply the splash screen theme

1. Navigate to the composeApp/src/androidMain/res/AndroidManifest.xml file
2. Change the theme of both the application and the first Activity to Theme.App.Splash.

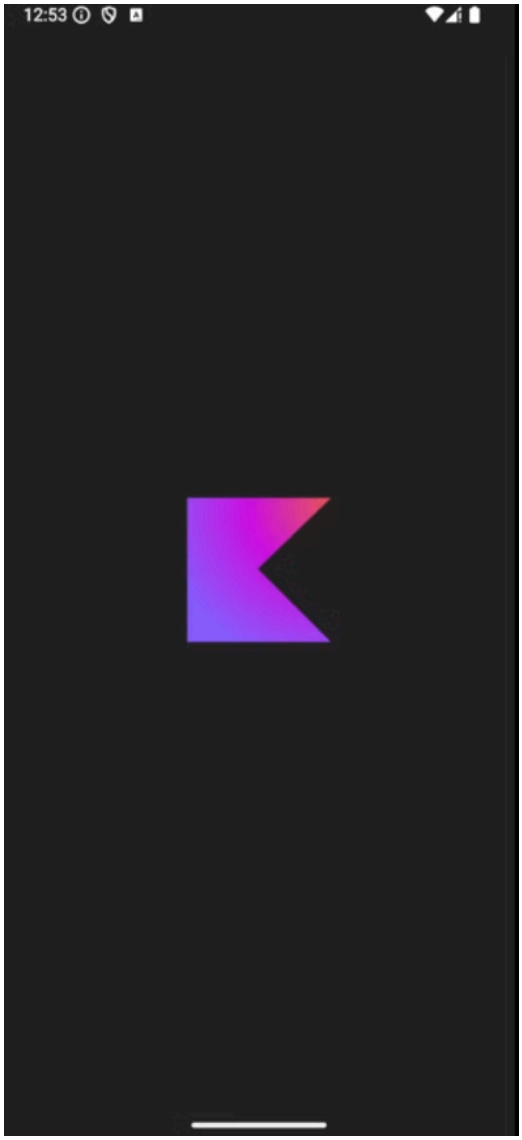
This could look as follows. In my example app, MainActivity is the only Activity in the app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application
        …
```

```
            android:theme="@style/Theme.App.Splash">
            <activity
                …
                android:name=".MainActivity"
                android:theme="@style/Theme.App.Splash">
                …
            </activity>
        </application>
    </manifest>
```

Install the splash screen

1. Open up the first activity in the application. In my example app, this is MainActivity.
2. Add a call to installSplashScreen() before calling setContent{...}



Run the application

1. Select composeApp in the Run Configurations menu, then click on the Run button.
2. The splash screen with the logo will show briefly, then disappear to show the first activity content.
   It might be necessary to change the scaling in the section *Making the vector drawable fit* to ensure no part of the icon is cut off.

[Optional] Make the splash screen stay conditionally

While the Android splash screen typically shows only a few seconds, it's possible to show it longer if the app isn't quite ready to show the first screen yet. For

example, if the first screen state depends on whether the app is in a logged-in state or not. We can keep the splash screen showing with the setKeepOnScreenCondition function.

A word of caution: a splash screen showing too long may create the impression that the app is slow or has frozen, so use this functionality with caution.

For the above example, we could write the following Activity code to simulate that the logged-in check takes one second.

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        var showSplashScreen = true
        var isLoggedIn = false

        lifecycleScope.launch {
            delay(1000) //Simulates checking if the user is logged in
            isLoggedIn = true
            showSplashScreen = false
        }

        installSplashScreen().apply {
            this.setKeepOnScreenCondition {
                showSplashScreen
            }
        }

        setContent {
            App(isLoggedIn = isLoggedIn)
        }
    }
}
```

Add an animated splash screen to the Android app

Adding a very short animation to the splash screen can add a moment of delight for your users. All the required steps from the *Adding a static splash screen to the Android app* section above are still required, there are just some additional modifications that need to be made to the app to make the splash screen animated.

The splash screen API currently doesn't allow for Jetpack Compose animations, but rather only XML animations. I find it much harder to get the type of animation I want from XML animations, but perhaps you are more experienced and can create something cooler (and again, quick!).

Set the logo's group name

1. Set the android:name attribute of the group created in the section *Making the vector drawable fit.* We will be using this name to refer to the group later on in the section.

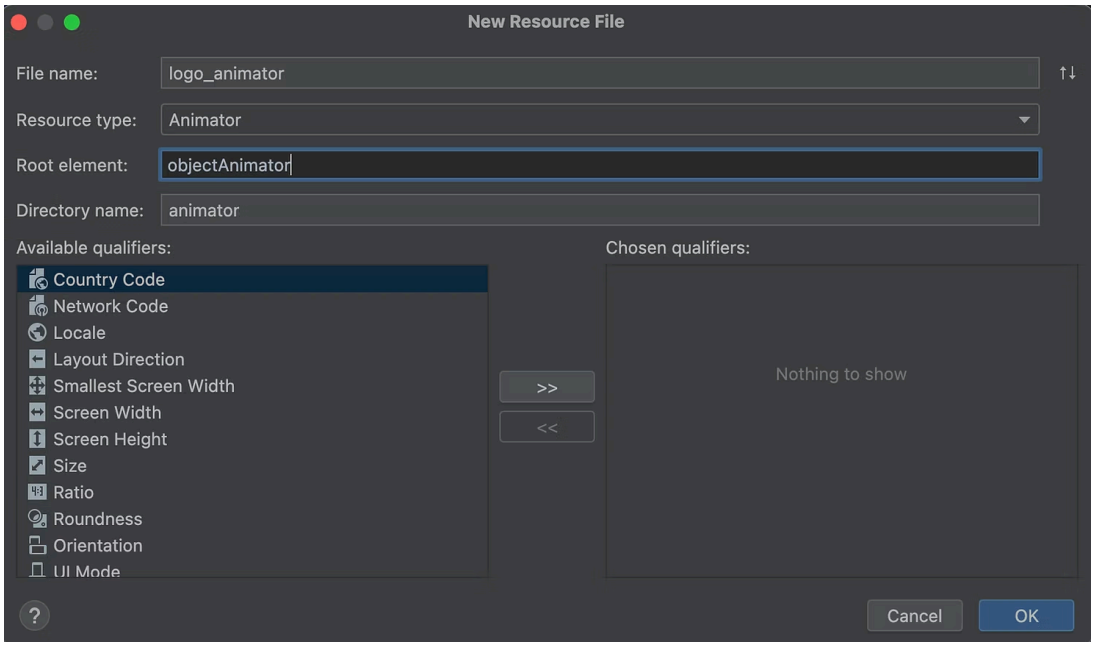
For example, we could set the name to logo_group.

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:aapt="http://schemas.android.com/aapt"
```

```
        …>
        <group
            …
        android:name="logo_group">

          <path … />

        </group>
    </vector>
```

Create the Animator

1. Right-click on the directory composeApp/src/androidMain/res
2. Select New > Android Resource File
3. In the New Resource File dialog, set the following:
    a. File name: Your choice, but I named it logo_animator.
    b. Resource type: Animator
    c. Root element: objectAnimator, as we'll be animating a group object.
    d. Directory name: animator
4. In the new file, you need to set up the animator with:
    a. A duration in milliseconds
    b. One or more propertyValueHolders.

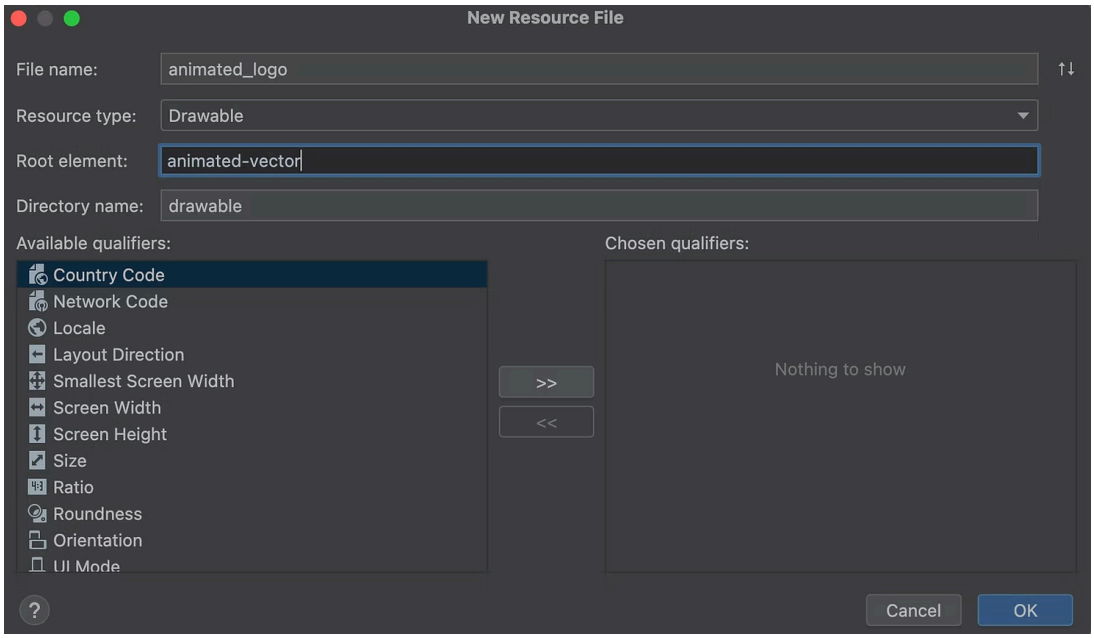


For example, below I have created an object animator that will animate for a duration of two seconds (android:duration), scaling the x-axis (scaleX) from 0.4 to 0.0 (so collapsing it), and the y-axis (scaleY) from 0.4 to 0.0.

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="2000">
    <propertyValuesHolder
        android:propertyName="scaleX"
        android:valueFrom="0.4"
        android:valueTo="0.0"
        android:valueType="floatType" />

    <propertyValuesHolder
        android:propertyName="scaleY"
        android:valueFrom="0.4"
        android:valueTo="0.0"
        android:valueType="floatType" />

</objectAnimator>
```

Create the animated vector drawable

1. Right-click on the directory composeApp/src/androidMain/res
2. Select New > Android Resource File
3. In the New Resource File dialog, set the following:
    a. File name: Your choice but I named it animated_logo
    b. Resource type: Drawable
    c. Root element: animated-vector
    d. Directory name: drawable
4. In the new file, you need to set up the animated_vector up with:
    a. A drawable, the image you want to animate
    b. A target, linking the animator and the group inside the drawable




```
<?xml version="1.0" encoding="utf-8"?>
<animated-vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/kotlin">

    <target
        android:animation="@animator/logo_animator"
        android:name="logo_group" />

</animated-vector>
```

Edit the splash screen theme

In the section *Creating a splash screen theme* we created a theme that now needs to be pointed to our animated vector drawable.

Let's now edit the splash screen theme:

1. Open composeApp/src/androidMain/res/values/splash_screen_theme.xml
2. Edit the resulting file to look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
        <style name="Theme.App.Splash" parent="Theme.SplashScreen">
            <item name="windowSplashScreenBackground">
                #202020
            </item>
            <item name="windowSplashScreenAnimatedIcon">
                @drawable/animated_logo
            </item>
            <item name="postSplashScreenTheme">
                @android:style/Theme.Material.NoActionBar
            </item>
        </style>
    </resources>
```

The theme will now display the animated logo instead of our static logo. When the app is run, we have a cute but simple animation to introduce our app.

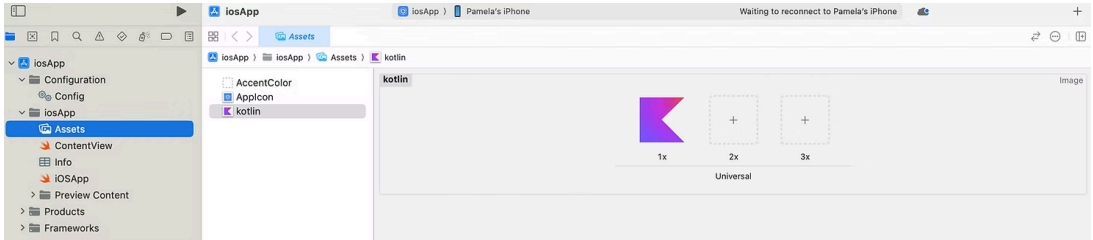
Add a static splash screen to the iOS app

Hop over to Xcode

1. In Android Studio, navigate to iosApp and right-click on iosApp.xcodeproj
2. Select Open In > Xcode
3. The project will now be correctly opened in Xcode

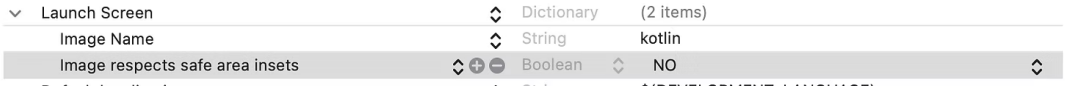
Add the image to your Xcode project

1. In Xcode, navigate to iosApp > iosApp and open up Assets
2. Drag and drop your logo into the white Assets view.



Create the Launch Screen

1. In the right side panel, navigate and click on the top-level iosApp (and make sure to be on the iosApp target)
2. In the middle panel, select Info
3. In the Customized iOS Target Properties table add a property Launch Screen
4. Add two properties underneath Launch Screen (there is autocomplete):
    a. Image Name, which is the image you added in the previous step, without the extension.
    b. Image respects safe area insets, which is whether to respect safe areas.



Run the iOS app to see the launch screen

If you have run the app before, there may have been some caching issues and the launch screen won't show. To fix this, a combination of the following might be necessary:

1. Navigate to the menu Product > Clean Build Folder
2. Hard-close the app already running. You need to swipe up from the bottom, and then close the specific app by swiping it up.
3. Delete the app from the device.

You can then run the app as usual and enjoy the very brief showing of the launch screen.

Add an animated splash screen to the iOS app

Instead of a static image, we can have an animation implemented with SwiftUI. From the previous section, it's necessary to Hop over to Xcode and Create the Launch Screen first.

Edit the code

1. In Xcode, navigate to the ContentView.swift file
2. Edit the ContentView struct to have two state variables:
   a. isHomeRootScreen: which is whether the current screen is the home/root screen (the Compose Multiplatform app) or the animated splash screen
   b. scaleAmount: with what scale amount the logo should be displayed

For example:

```
struct ContentView: View {

    @State var scaleAmount = 1.0

    @State var isHomeRootScreen = false

    …

}
```

In the body of the ContentView struct, edit the code to introduce a ZStack. The content of the ZStack is conditional based on the state of isHomeRootScreen: either the Compose Multiplatform app or a Image with logo.

```
var body: some View {

        ZStack {

            if isHomeRootScreen {

                ComposeView()

                    .ignoresSafeArea(.keyboard)

            } else {

                Image(.kotlin)

            }

        }

       .ignoresSafeArea(( isHomeRootScreen ? .keyboard : .all))

}
```

Configure the logo Image as follows:

```
Image(.kotlin)

    .resizable()

    .aspectRatio(contentMode: .fit)

    .scaleEffect(scaleAmount)

    .frame(width:60)
```

Add the animation to the Image view onAppear. The animation will shrink the image to nothing in two seconds.

```
Image(.kotlin)

    …
```

```
        .onAppear() {
            withAnimation(.easeIn(duration: 2)){
                scaleAmount = 0.0
            }
       }
```

Also navigate to the home/root screen by setting the isHomeRootScreen variable to true. This needs to be delayed by two seconds so the animation completes.

```
    DispatchQueue.main.asyncAfter(deadline: .now() + 2, execute: {
        isHomeRootScreen = true
    })
```

Ultimately, the ContentView struct will look as follows:

```
    struct ContentView: View {
        @State var scaleAmount = 1.0
        @State var isHomeRootScreen = false

        var body: some View {
            ZStack {
                if isHomeRootScreen {
                    ComposeView()
                } else {
                    Image(.kotlin)
                        .resizable()
                        .aspectRatio(contentMode: .fit)
                        .scaleEffect(scaleAmount)
                        .frame(width:60)
                        .onAppear() {
                            withAnimation(.easeIn(duration: 2)){
                                scaleAmount = 0.0
                            }
                            DispatchQueue.main.asyncAfter(deadline: .now() + 2, execute: {
```

```
                                isHomeRootScreen = true

                            })

                        }

                    }

                }.ignoresSafeArea(( isHomeRootScreen ? .keyboard : .all))

            }

        }
```


References (with thanks!)

Splash screens on developer.android.com

Launch screens in Xcode: All the options explained by Antoine van der Lee

Create a Splash Screen in Compose Multiplatform for iOS & Android - KMP for Beginners by Philipp Lackner

How to Build an Animated Splash Screen on Android - The Full Guide by Philipp Lackner

App Icon & Splash Screen for iOS and Android (Dark/Light) - Compose Multiplatform by Stefan Jovanović

Create an Animated Launch Screen like Twitter - SwiftUI by Wes Chua

---

With thanks to Márton Braun and Konstantin Tskhovrebov for their help and reviews.