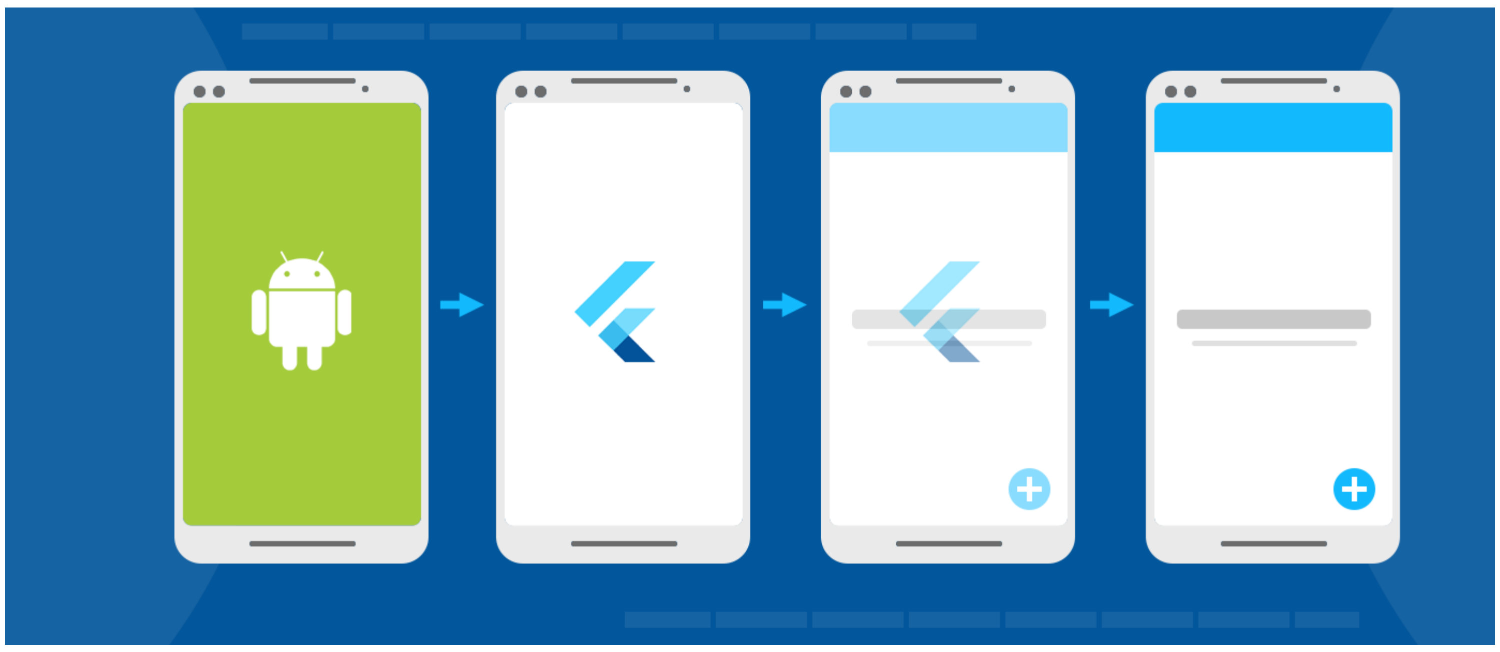
Get started
Samples & tutorials
Development
User interface
Introduction to widgets
Building layouts
Splash screens
Android Splash Screens
Adding interactivity
Assets and images
Navigation & routing
Animations
Advanced UI
Widget catalog
Data & backend
Accessibility & internationalization
Platform integration
Packages & plugins
Add Flutter to existing app
Tools & techniques
Migration notes
Testing & debugging
Performance & optimization
Deployment
Resources
Reference
Widget index
API reference
Package site

Adding a splash screen and launch screen to an Android app

Contents



The beginning of a Flutter experience requires a brief wait while Dart initializes. Additionally, a full Flutter app requires standard Android app initialization time. Flutter supports the display of a launch screen while your Android app initializes, and also support the display of a splash screen while your Flutter experience initializes. This guide teaches you how to use launch screens and spl screens in an Android app with Flutter.

Note: Strategies are available to minimize wait time related to Flutter initialization. Consider pre-warming a `FlutterEngine` and reusing a FlutterEngine throughout your app to avoid most wait time.

Android launch screen

Every Android app requires initialization time while the operating system sets up the app's process. Android provides the concept launch screen to display a `Drawable` while the app is initializing.

Flutter provides support for displaying an Android launch screen before showing a `FlutterActivity`. The instructions to display Android launch screen are discussed in the next sections.

Define a launch theme

In `styles.xml`, define a theme whose `windowBackground` is set to the `Drawable` that should be displayed as the launch screen.

```
<style name="LaunchTheme" parent="@android:style/Theme.Black.NoTitleBar">
    <item name="android:windowBackground">@drawable/launch_background</item>
</style>
```

Note: The default Flutter project template includes a definition of a launch theme and a launch background.

Define a normal theme

In `styles.xml`, define a normal theme to be applied to `FlutterActivity` after the launch screen is gone. The normal theme background only shows for a very brief moment after the splash screen disappears, and during orientation change and `Activity` restoration. Therefore, it's recommended that the normal theme use a solid background color that looks similar to the primary background color of the Flutter UI.

```
<style name="NormalTheme" parent="@android:style/Theme.Black.NoTitleBar">
    <item name="android:windowBackground">@drawable/normal_background</item>
</style>
```

Setup FlutterActivity in AndroidManifest.xml

In `AndroidManifest.xml`, set the `theme` of `FlutterActivity` to the launch theme. Then, add a metadata element to the desired `FlutterActivity` to instruct Flutter to switch from the launch theme to the normal theme at the appropriate time.

```
<activity
    android:name=".MyActivity"
    android:theme="@style/LaunchTheme"
    // ...
    >
    <meta-data
        android:name="io.flutter.embedding.android.NormalTheme"
        android:resource="@style/NormalTheme"
        />
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

The Android app now displays the desired launch screen while the app initializes.

Flutter splash screen

Each Flutter experience in an app requires a few moments to initialize the Dart isolate that runs the code. This means a user momentarily sees a blank screen until Flutter renders its first frame. Flutter supports an improved user experience by displaying a Android `View` as a splash screen while Flutter initializes.

Flutter supports two options for a splash screen. The first option is to display a `Drawable` of your choice, which fades out after the initialization is complete. The other option is to provide a custom `SplashScreen`, which is capable of displaying any Android `View` content that you want.

Showing a Drawable splash screen

A `Drawable` splash screen can be configured for a `FlutterActivity`, `FlutterFragment`, or `FlutterView`.

In a FlutterActivity

To display a `Drawable` as a Flutter splash screen in a `FlutterActivity`, add the following metadata to the associated `FlutterActivity` in `AndroidManifest.xml`.

```
<meta-data
    android:name="io.flutter.embedding.android.SplashScreenDrawable"
    android:resource="@drawable/my_splash"
    />
```

To display a splash screen with the same visual as a launch screen, reference the same `@drawable/launch_background` in the `io.flutter.embedding.android.SplashScreenDrawable` `meta-data`.

In a FlutterFragment

To display a `Drawable` as a Flutter splash screen in a `FlutterFragment`, make `FlutterFragment` a subclass and override `provideSplashScreen()`.

```
public class MyFlutterFragment extends FlutterFragment {
    @Override
    protected SplashScreen provideSplashScreen() {
        // Load the splash Drawable.
        Drawable splash = getResources().getDrawable(R.drawable.my_splash);

        // Construct a DrawableSplashScreen with the loaded splash Drawable and
        // return it.
        return new DrawableSplashScreen(splash);
    }
}
```

Creating a custom SplashScreen

Splash screens are a great branding opportunity. Because of that, many teams implement unique, highly customized splash experiences. To facilitate this, Flutter allows you to display an arbitrary Android `View` as a splash screen, and even allows you to control how that `View` transitions to Flutter after Flutter renders its first frame.

Implement a custom splash View

First, define the custom `View` that should be displayed as the splash screen.

This `View` could display anything, from a simple solid color to an animation. An example isn't provided because there are too man options.

Implement the SplashScreen interface

With a custom `View` defined, implement the `SplashScreen` interface.

This guide shows two approaches to a `SplashScreen` implementation. First, the following is an example of a `SplashScreen` that l no visual state and no transition animation.

```
public class SimpleSplashScreen implements SplashScreen {
    @Override
    @Nullable
    public View createSplashView(
      @NonNull Context context,
      @Nullable Bundle savedInstanceState
    ) {
        // Return a new MySplashView without saving a reference, because it
        // has no state that needs to be tracked or controlled.
        return new MySplashView(context);
    }

    @Override
    public void transitionToFlutter(@NonNull Runnable onTransitionComplete) {
        // Immediately invoke onTransitionComplete because this SplashScreen
        // doesn't display a transition animation.
        //
        // Every SplashScreen *MUST* invoke onTransitionComplete at some point
        // for the splash system to work correctly.
        onTransitionComplete.run();
    }
}
```

The second example is a bit more sophisticated. In this example, the custom `SplashScreen` keeps a reference to its custom `View` instructs the custom `View` to transition away, passing the `onTransitionComplete` callback to the custom `View` to invoke.

```
public class SplashScreenWithTransition implements SplashScreen {
    private MySplashView mySplashView;

    @Override
    @Nullable
    public View createSplashView(
      @NonNull Context context,
      @Nullable Bundle savedInstanceState
    ) {
        // A reference to the MySplashView is retained so that it can be told
        // to transition away at the appropriate time.
        mySplashView = new MySplashView(context);
        return mySplashView;
    }

    @Override
    public void transitionToFlutter(@NonNull Runnable onTransitionComplete) {
        // Instruct MySplashView to animate away in whatever manner it wants.
        // The onTransitionComplete Runnable is passed to the MySplashView to be
        // invoked when the transition animation is complete.
        mySplashView.animateAway(onTransitionComplete);
    }
}
```

With custom splash screens, the sky is the limit. In fact, you could create a splash screen that shows an animated sky! Have fun this flexible splash system, and share your creations with the community!