FLUTTER TUTORIALS

BY: **TECMAN**

TECMAN Lesson 7

Adding Assets and Images

Flutter apps can include both code and *assets* (sometimes called resources). An asset is a file that is bundled and deployed with your app, and is accessible at runtime. Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP).

TECMAN LIMITED 2019

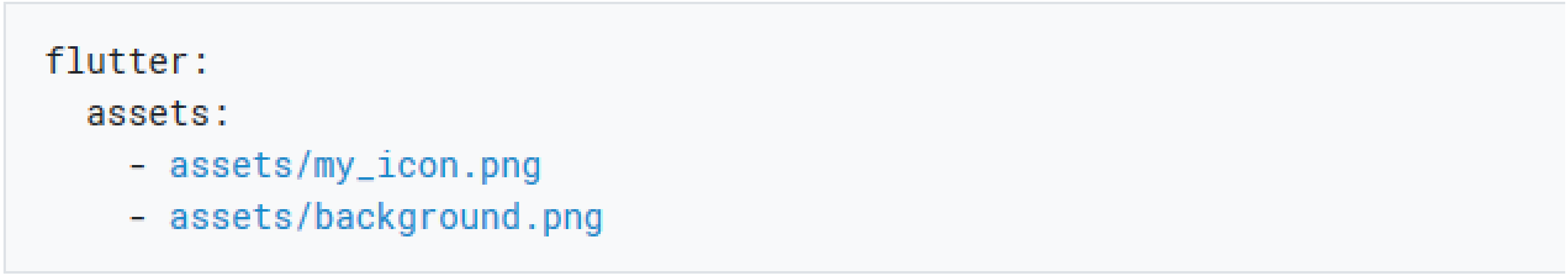
Specifying assets

- Flutter uses the pubspec.yaml file, located at the root of your project, to identify assets required by an app.

TECMAN LIMITED 2019

Specifying assets

- Here is an example:

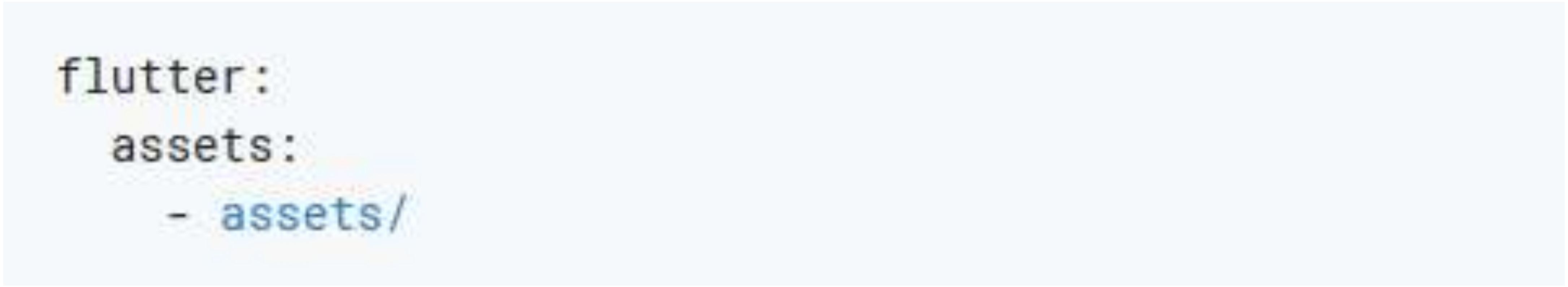


```
flutter:
  assets:
    - assets/my_icon.png
    - assets/background.png
```

TECMAN LIMITED 2019

Specifying assets

- To include all assets under a directory, specify the directory name with the / character at the end:



```
flutter:
  assets:
    - assets/
```

TECMAN LIMITED 2019

Specifying assets

- Note that only files located directly in the directory will be included; to add files located in subdirectories, create an entry per directory.

Asset bundling

The assets subsection of the flutter section specifies files that should be included with the app. Each asset is identified by an explicit path (relative to the pubspec.yaml file) where the asset file is located. The order in which the assets are declared does not matter. The actual directory used (assets in this case) does not matter.

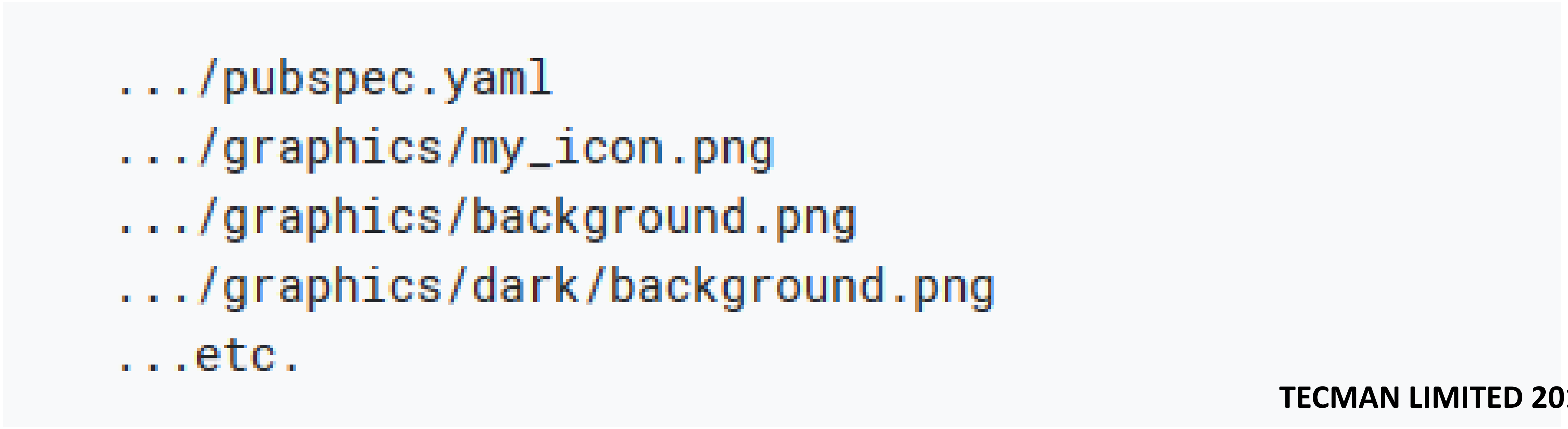
During a build, Flutter places assets into a special archive called the *asset bundle*, which apps can read from at runtime.

Asset variants

- The build process supports the notion of asset variants: different versions of an asset that might be displayed in different contexts. When an asset's path is specified in the assets section of pubspec.yaml, the build process looks for any files with the same name in adjacent subdirectories. Such files are then included in the asset bundle along with the specified asset.

Asset variants

- For example, if you have the following files in your application directory:

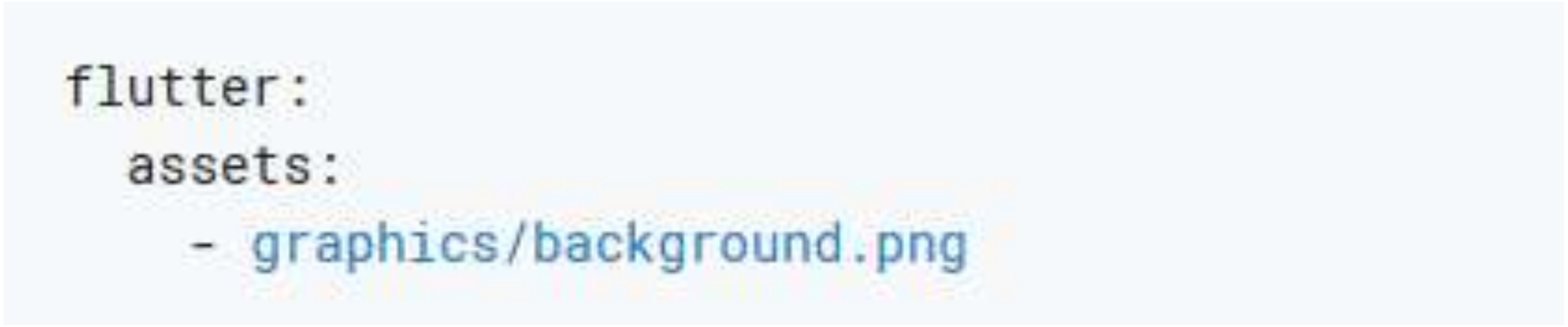


```
.../pubspec.yaml
.../graphics/my_icon.png
.../graphics/background.png
.../graphics/dark/background.png
...etc.
```

TECMAN LIMITED 2019

Asset variants

- …and your pubspec.yaml file contains:



```
flutter:
  assets:
    - graphics/background.png
```

TECMAN LIMITED 2019

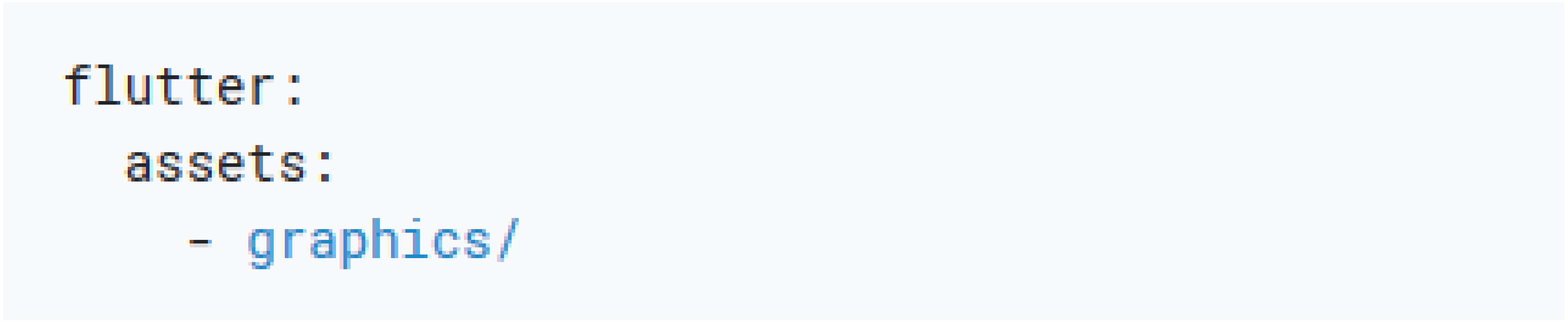
Asset variants

…then both graphics/background.png and graphics/dark/background.png will be included in your asset bundle. The former is considered the *main asset*, while the latter is considered a *variant*.

If on the other hand the graphics directory is specified:

TECMAN LIMITED 2019

Asset variants



```yaml
flutter:
  assets:
    - graphics/
```

Asset variants

… then graphics/my_icon.png, graphics/background.png and graphics/dark/background.png will be included.

Flutter uses asset variants when choosing resolution appropriate images; see below. In the future, this mechanism might be extended to include variants for different locales or regions, reading directions, etc.

Loading assets

Your app can access its assets through an AssetBundle object.

The two main methods on an asset bundle allow you to load a string/text asset (loadString) or an image/binary asset (load) out of the bundle, given a logical key. The logical key maps to the path to the asset specified in the pubspec.yaml file at build time.

Loading text assets

Each Flutter app has a rootBundle object for easy access to the main asset bundle. It is possible to load assets directly using the rootBundle global static from package:flutter/services.dart.

However, it’s recommended to obtain the AssetBundle for the current BuildContext using DefaultAssetBundle. Rather than the default asset bundle that was built with the app, this approach enables a parent widget to substitute a different AssetBundle at run time, which can be useful for localization or testing scenarios.

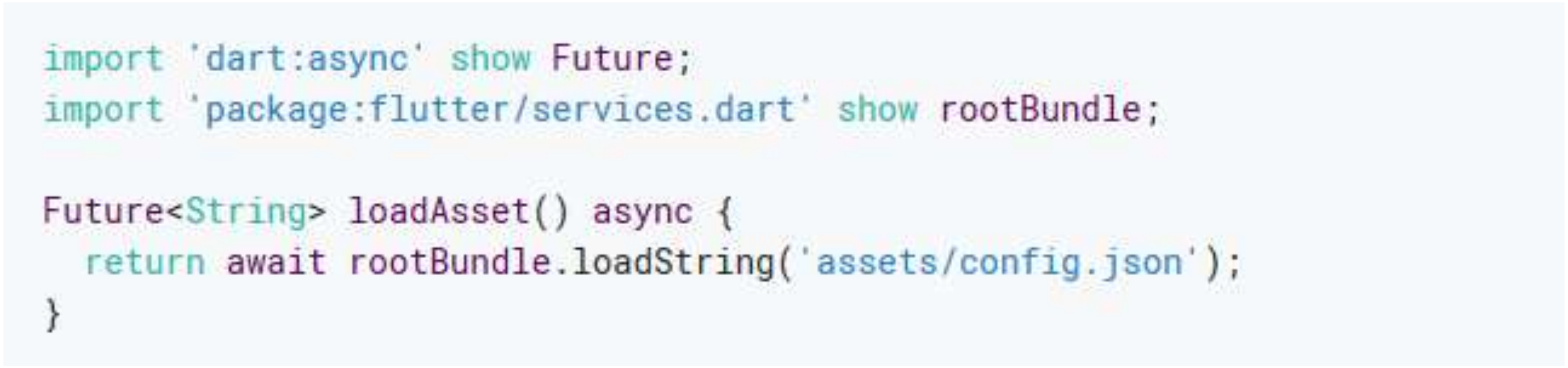
Loading text assets

Typically, you’ll use DefaultAssetBundle.of() to indirectly load an asset, for example a JSON file, from the app’s runtime rootBundle.

Outside of a Widget context, or when a handle to an AssetBundle is not available, you can use rootBundle to directly load such assets, for example:

TECMAN LIMITED 2019

Loading text assets



```
import 'dart:async' show Future;
import 'package:flutter/services.dart' show rootBundle;

Future<String> loadAsset() async {
  return await rootBundle.loadString('assets/config.json');
}
```

TECMAN LIMITED 2019

Loading images

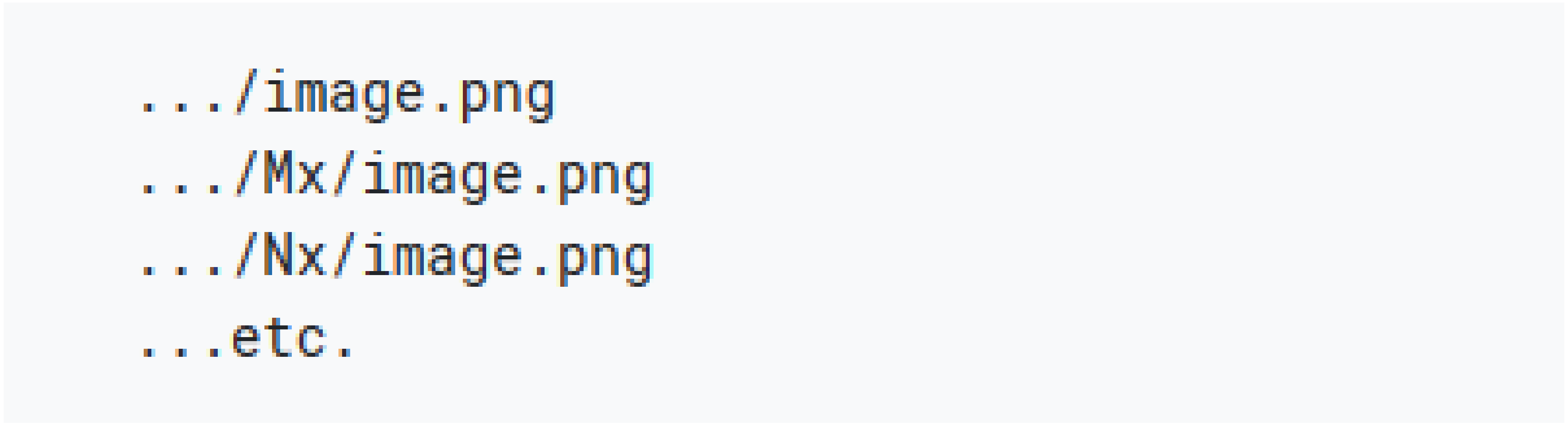
- Flutter can load resolution-appropriate images for the current device pixel ratio

Loading images

Declaring resolution-aware image assets

AssetImage understands how to map a logical requested asset onto one that most closely matches the current device pixel ratio. In order for this mapping to work, assets should be arranged according to a particular directory structure:

Loading images



```
.../image.png
.../Mx/image.png
.../Nx/image.png
...etc.
```

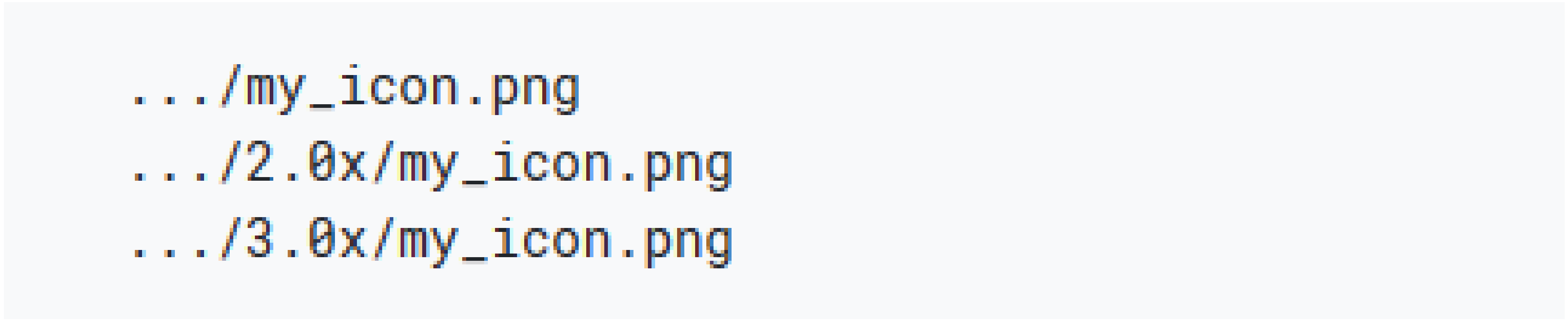
TECMAN LIMITED 2019

Loading images

…where *M* and *N* are numeric identifiers that correspond to the nominal resolution of the images contained within, in other words, they specify the device pixel ratio that the images are intended for.

The main asset is assumed to correspond to a resolution of 1.0. For example, consider the following asset layout for an image named my_icon.png:

Loading images



```
.../my_icon.png
.../2.0x/my_icon.png
.../3.0x/my_icon.png
```

Loading images

On devices with a device pixel ratio of 1.8, the asset .../2.0x/my_icon.png would be chosen. For a device pixel ratio of 2.7, the asset .../3.0x/my_icon.png would be chosen.

If the width and height of the rendered image are not specified on the Image widget, the nominal resolution is used to scale the asset so that it will occupy the same amount of screen space as the main asset would have, just with a higher resolution. That is, if .../my_icon.png is 72px by 72px, then .../3.0x/my_icon.png should be 216px by 216px; but they both will render into 72px by 72px (in logical pixels) if width and height are not specified.

Loading images

- Each entry in the asset section of the pubspec.yaml should correspond to a real file, with the exception of the main asset entry. If the main asset entry does not correspond to a real file, then the asset with the lowest resolution will be used as the fallback for devices with device pixel ratios below that resolution. The entry should still be included in the pubspec.yaml manifest, however.

TECMAN LIMITED 2019

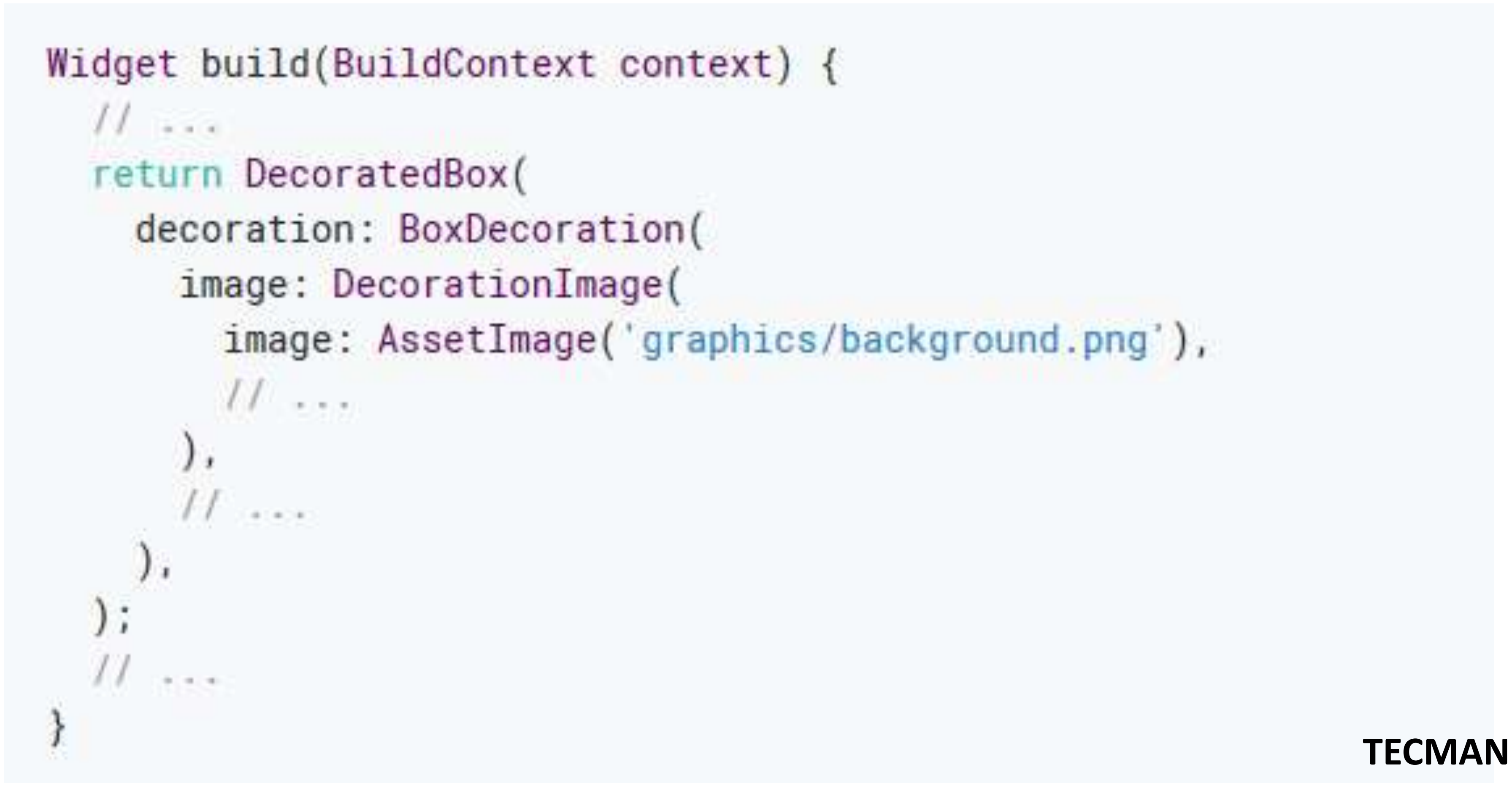
Loading images

To load an image, use the AssetImage class in a widget’s build method.

For example, your app can load the background image from the asset declarations above:

TECMAN LIMITED 2019

Loading images



```
Widget build(BuildContext context) {
  // ...
  return DecoratedBox(
    decoration: BoxDecoration(
      image: DecorationImage(
        image: AssetImage('graphics/background.png'),
        // ...
      ),
      // ...
    ),
  );
  // ...
}
```

Loading images

- Anything using the default asset bundle will inherit resolution awareness when loading images. (If you work with some of the lower level classes, like ImageStream or ImageCache, you’ll also notice parameters related to scale.)

Asset images in package dependencies

To load an image from a package dependency, the package argument must be provided to AssetImage.

For instance, suppose your application depends on a package called my_icons, which has the following directory structure:

TECMAN LIMITED 2019

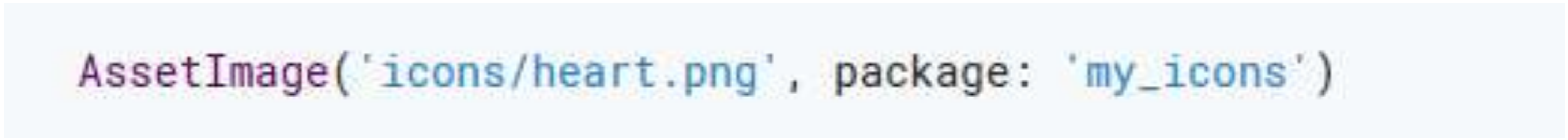
Asset images in package dependencies

```
.../pubspec.yaml
.../icons/heart.png
.../icons/1.5x/heart.png
.../icons/2.0x/heart.png
...etc.
```

TECMAN LIMITED 2019

Asset images in package dependencies

- Then to load the image, use:



```
AssetImage('icons/heart.png', package: 'my_icons')
```

- Assets used by the package itself should also be fetched using the package argument as above.

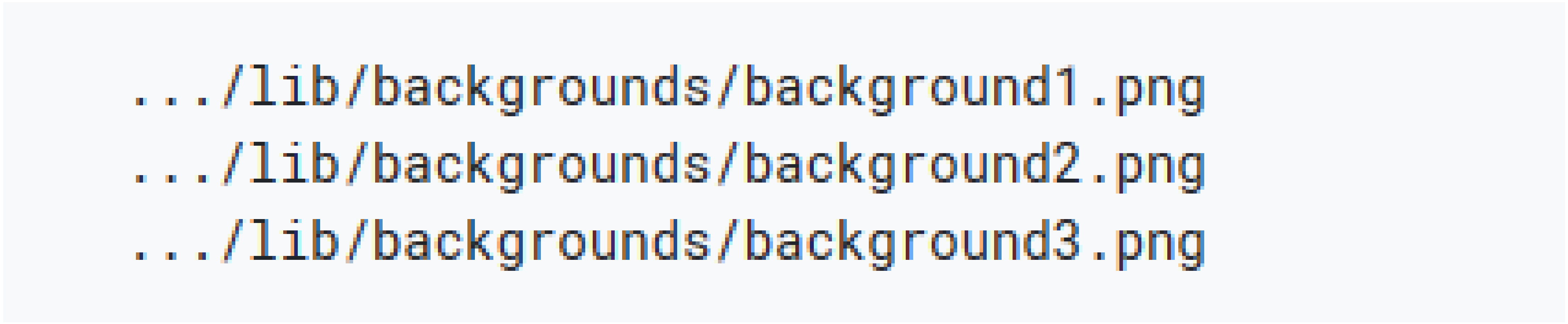
Bundling of package assets

If the desired asset is specified in the pubspec.yaml file of the package, it is bundled automatically with the application. In particular, assets used by the package itself must be specified in its pubspec.yaml.

A package can also choose to have assets in its lib/ folder that are not specified in its pubspec.yaml file. In this case, for those images to be bundled, the application has to specify which ones to include in its pubspec.yaml. For instance, a package named fancy_backgrounds could have the following files:

TECMAN LIMITED 2019

Bundling of package assets



```
.../lib/backgrounds/background1.png
.../lib/backgrounds/background2.png
.../lib/backgrounds/background3.png
```

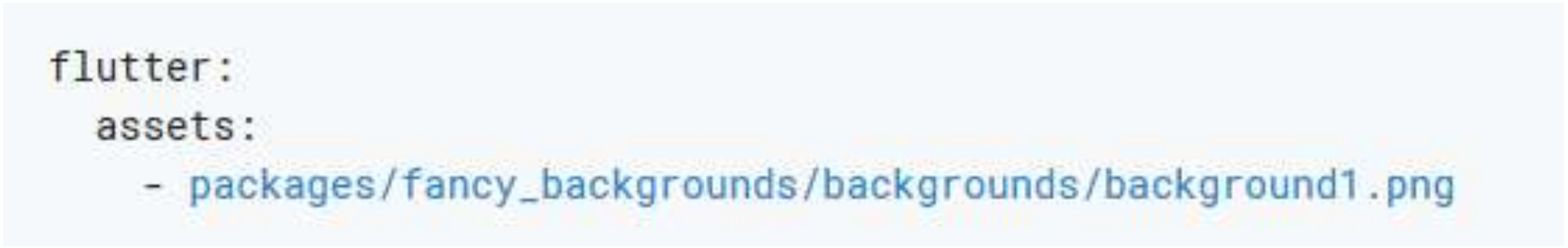
TECMAN LIMITED 2019

Bundling of package assets

- To include, say, the first image, the pubspec.yaml of the application should specify it in the assets section:

TECMAN LIMITED 20191

Bundling of package assets



```
flutter:
  assets:
    - packages/fancy_backgrounds/backgrounds/background1.png
```

- The lib/ is implied, so it should not be included in the asset path.

TECMAN LIMITED 2019

Sharing assets with the underlying platform

- Flutter assets are readily available to platform code via AssetManager on Android and NSBundle on iOS.

Android

On Android the assets are available via the AssetManager API. The lookup key used in for instance openFd is obtained from lookupKeyForAsset on PluginRegistry.Registrar or getLookupKeyForAsset on FlutterView. PluginRegistry.Registrar is available when developing a plugin while FlutterView would be the choice when developing an app including a platform view.

As an example, suppose you have specified this in your pubspec.yaml

TECMAN LIMITED 2019

Android

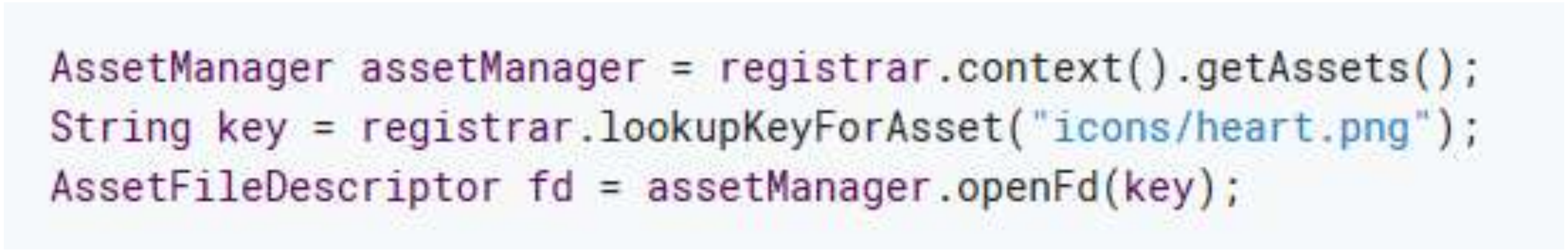
```
flutter:
  assets:
    - icons/heart.png
```

- reflecting the following structure in your Flutter app.

```
  .../pubspec.yaml
  .../icons/heart.png
  ...etc.
```

Android

- To access icons/heart.png from your Java plugin code you would do;



```
AssetManager assetManager = registrar.context().getAssets();
String key = registrar.lookupKeyForAsset("icons/heart.png");
AssetFileDescriptor fd = assetManager.openFd(key);
```

TECMAN LIMITED 2019

iOS

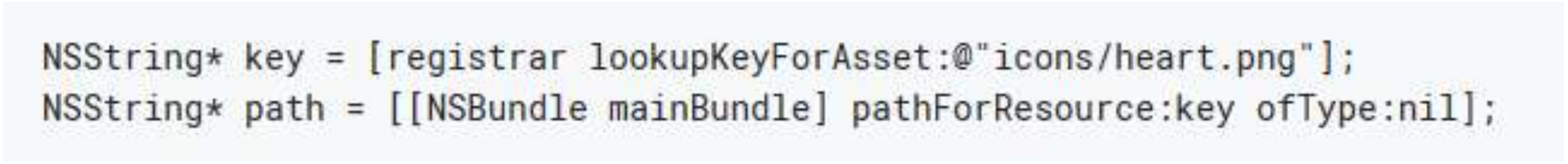
- On iOS the assets are available via the mainBundle. The lookup key used in for instance pathForResource:ofType: is obtained from lookupKeyForAsset or lookupKeyForAsset:fromPackage: on FlutterPluginRegistrar or lookupKeyForAsset: or lookupKeyForAsset:fromPackage: on FlutterViewController. FlutterPluginRegistrar is available when developing a plugin while FlutterViewController would be the choice when developing an app including a platform view.

TECMAN LIMITED 2019

iOS

As an example, suppose you have the Flutter setting from above.

To access icons/heart.png from your Objective-C plugin code you would do;



```
NSString* key = [registrar lookupKeyForAsset:@"icons/heart.png"];
NSString* path = [[NSBundle mainBundle] pathForResource:key ofType:nil];
```

Platform assets

- There will also be occasions to work with assets in the platform projects directly. Below are two common cases where assets are used before the Flutter framework is loaded and running.

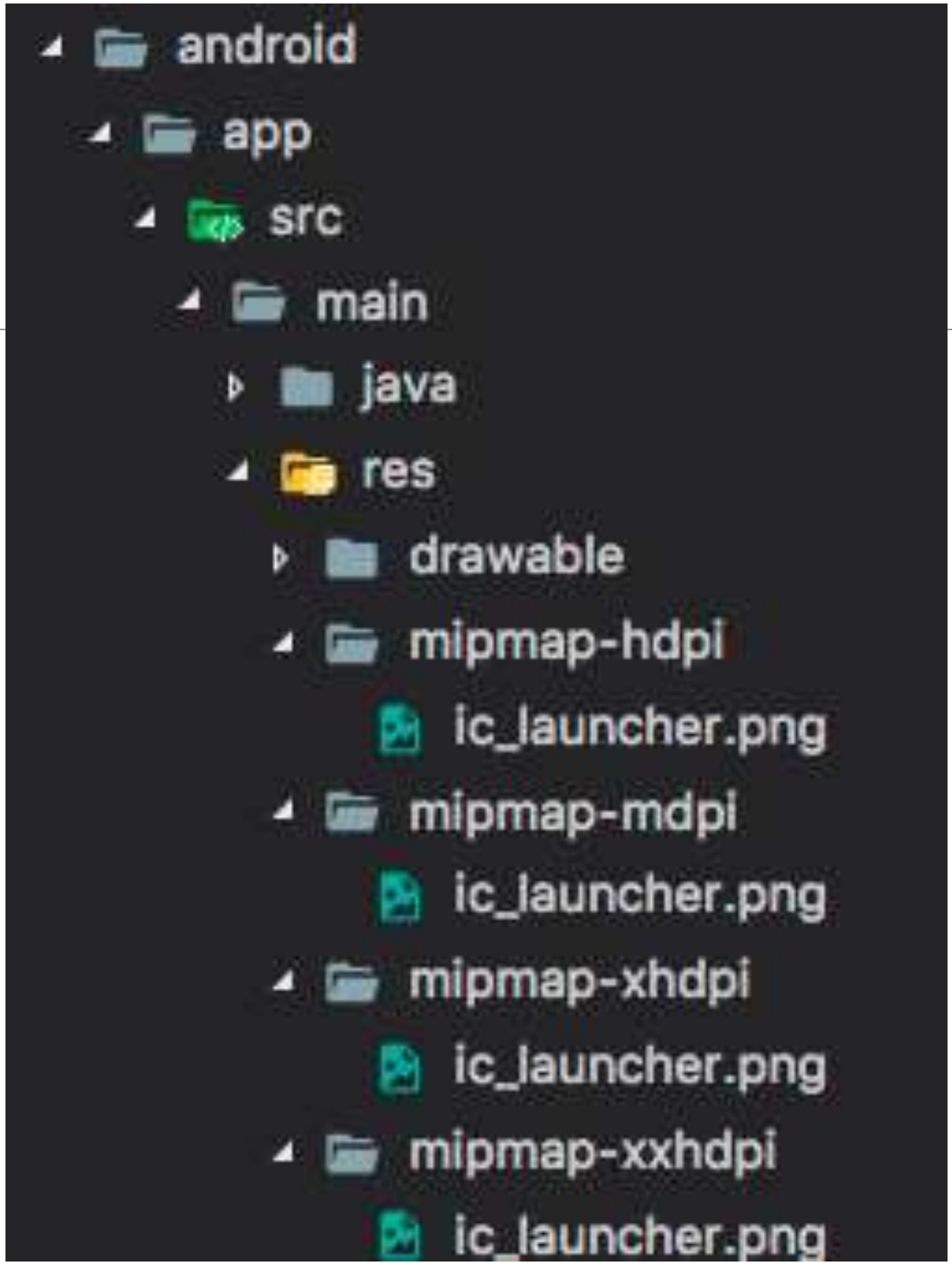
Updating the app icon

- Updating your Flutter application's launch icon works the same way as updating launch icons in native Android or iOS applications.

TECMAN LIMITED 2019

Android

In your Flutter project’s root directory, navigate to .../android/app/src/main/res. The various bitmap resource folders such as mipmap-hdpi already contain placeholder images named ic_launcher.png. Simply replace them with your desired assets respecting the recommended icon size per screen density as indicated by the Android Developer Guide.



```
android
  app
    src
      main
        java
        res
          drawable
          mipmap-hdpi
            ic_launcher.png
          mipmap-mdpi
            ic_launcher.png
          mipmap-xhdpi
            ic_launcher.png
          mipmap-xxhdpi
            ic_launcher.png
```

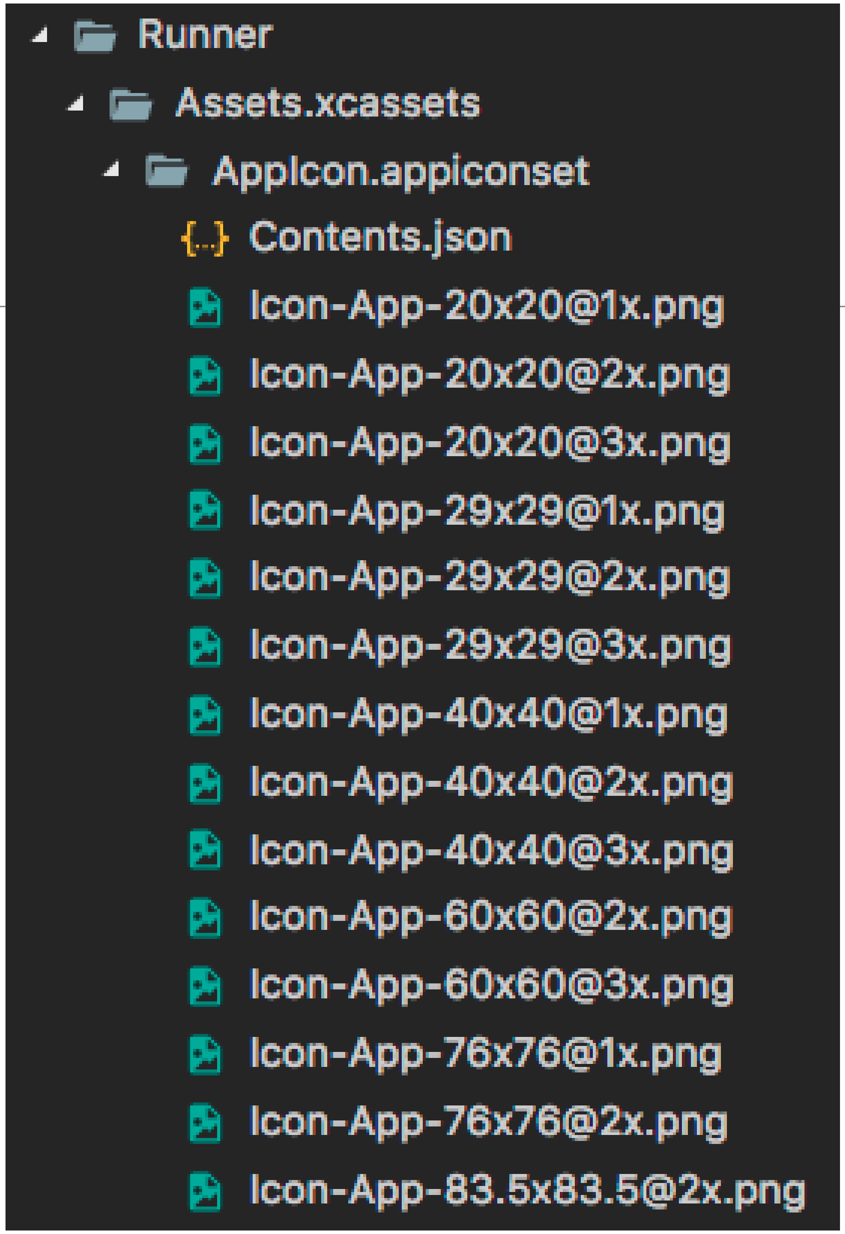
TECMAN LIMITED 2019

Android

- **Note:** If you rename the .png files, you must also match the new name in your AndroidManifest.xml's <application> tag's android:icon attribute.

iOS

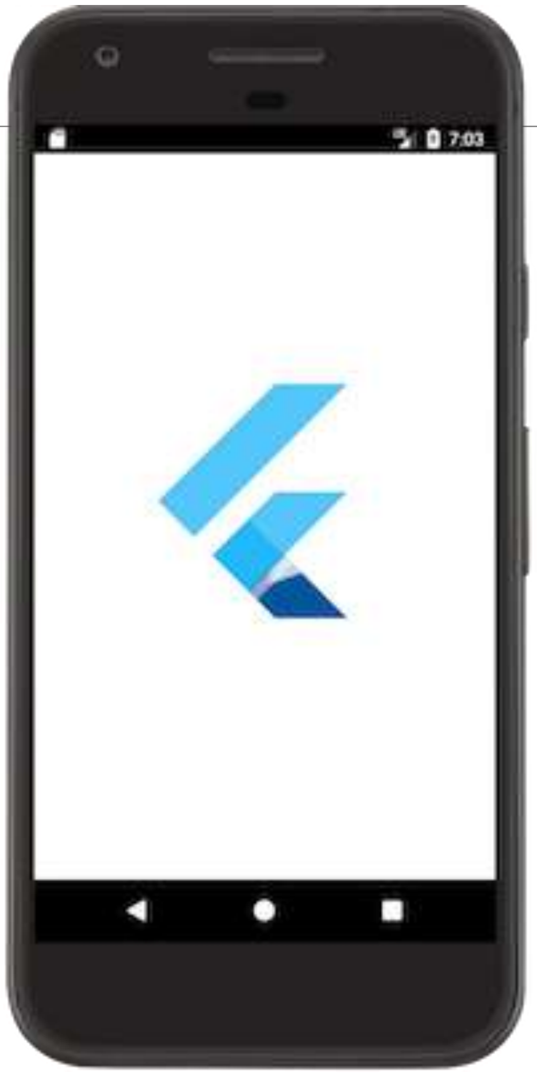
- In your Flutter project’s root directory, navigate to .../ios/Runner. The directory Assets.xcassets/AppIcon.appiconset already contains placeholder images. Simply replace them with the appropriately sized images as indicated by their filename as dictated by the Apple Human Interface Guidelines. Keep the original file names.



```
Runner
  Assets.xcassets
    AppIcon.appiconset
      Contents.json
      Icon-App-20x20@1x.png
      Icon-App-20x20@2x.png
      Icon-App-20x20@3x.png
      Icon-App-29x29@1x.png
      Icon-App-29x29@2x.png
      Icon-App-29x29@3x.png
      Icon-App-40x40@1x.png
      Icon-App-40x40@2x.png
      Icon-App-40x40@3x.png
      Icon-App-60x60@2x.png
      Icon-App-60x60@3x.png
      Icon-App-76x76@1x.png
      Icon-App-76x76@2x.png
      Icon-App-83.5x83.5@2x.png
```

TECMAN LIMITED 2019

Updating the launch screen



TECMAN LIMITED 2019

Updating the launch screen

- Flutter also uses native platform mechanisms to draw transitional launch screens to your Flutter app while the Flutter framework loads. This launch screen will persist until Flutter renders the first frame of your application.

TECMAN LIMITED 2019

Updating the launch screen

- **Note:** This implies that if you don’t call runApp() in the main() function of your app (or more specifically, if you don’t call window.render() in response to window.onDrawFrame), the launch screen will persist forever.

TECMAN LIMITED 2019

Android

To add a “splash screen” to your Flutter application, navigate to .../android/app/src/main. In res/drawable/launch_background.xml, You can use this layer list drawable XML to customize the look of your launch screen. The existing template provides an example for adding a image to to the middle of a white splash screen in commented code. You can uncomment it or use other drawables to achieve the intended effect.

TECMAN LIMITED 2019

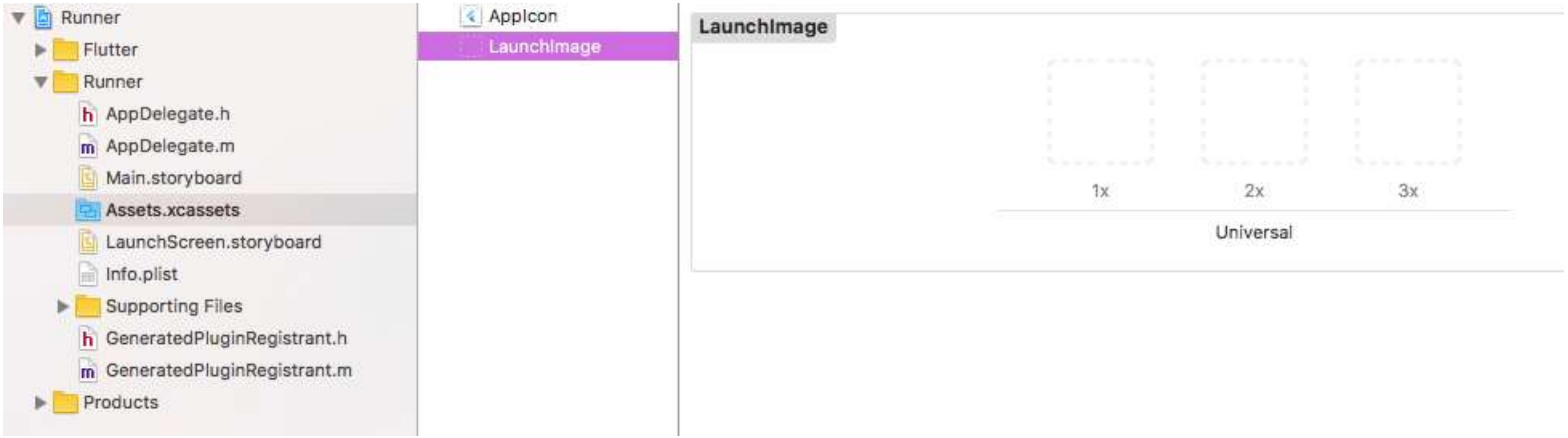
iOS

To add an image to the center of your “splash screen”, navigate to .../ios/Runner. In, Assets.xcassets/LaunchImage.imageset, drop in images named LaunchImage.png, LaunchImage@2x.png, LaunchImage@3x.png. If you use different filenames, you’ll also have to update the Contents.json file in the same directory.

iOS

- You can also fully customize your launch screen storyboard in Xcode by opening .../ios/Runner.xcworkspace. Navigate to Runner/Runner in the Project Navigator and drop in images by opening Assets.xcassets or do any customization using the Interface Builder in LaunchScreen.storyboard.

iOS



TECMAN LIMITED 2019

THANK YOU

TECMAN LIMITED 2019