Working with Custom Fonts in Flutter App:

- In-order to use the custom fonts in flutter application first we need to have a font that can be downloaded through the different sources.

- Download one of the fonts that you need: https://fonts.google.com/

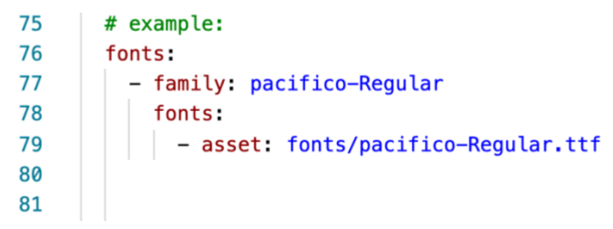
- Extract the downloaded zip file.

- Now we must go to the application where you want to use it. See the project structure and in project structure create a new folder **'fonts'.**

- Drag the font file from the extracted folder in to the 'fonts' folder.

- After completing the above steps, go to the **'pubspec.yaml'** and open it.

- Go to the:



```yaml
  # example:
  fonts:
    - family: pacifico-Regular
      fonts:
        - asset: fonts/pacifico-Regular.ttf


```

-

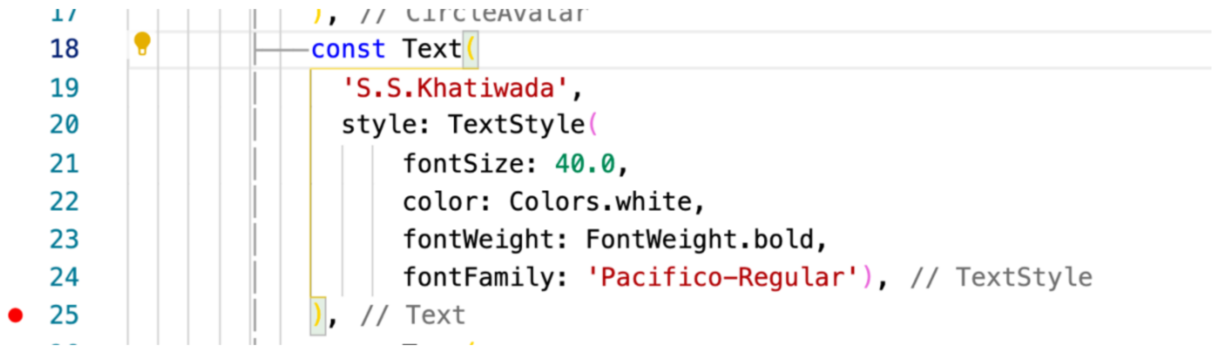
- if the 'fonts' is comment, uncomment them all.

- Provide the font name in **family** that you want to use.

- Provide font path of font using extension in **asset.**

- After doing all these, go to the terminal and write the command 'flutter pub get' or just click the download button shown in the right corner of '**pubspec.yaml'** file.

Now you can use this font in your application:



```
), // CircleAvatar
const Text(
  'S.S.Khatiwada',
  style: TextStyle(
      fontSize: 40.0,
      color: Colors.white,
      fontWeight: FontWeight.bold,
      fontFamily: 'Pacifico-Regular'), // TextStyle
), // Text
```

Working with Image in Flutter:

Image class has constructors:

1. **Image.asset** - To display image from assets bundle
2. **Image.file** - To display image from a file
3. **Image.memory** - To display image from Uint8List
4. **Image.network** - To display image from a URL

❖ Image.asset

Flutter is an open-source, cross-platform UI development kit developed by Google. It is gaining popularity these days, as the app made in flutter can run on various devices regardless of their platform. It is majorly used to develop applications for Android and iOS, as a single app made in flutter can work efficiently on both platforms.

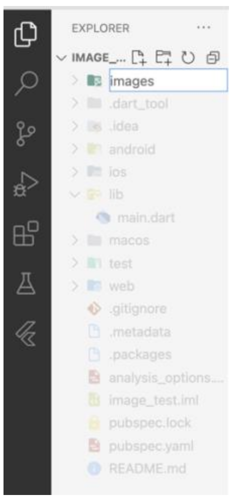
In this article, we will learn how to add images in the flutter app. A flutter app when built has both assets (resources) and code. Assets are available and deployed during runtime. The asset is a file that can include static data, configuration files, icons, and images. The Flutter app supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF, BMP, and WBMP.

Syntax:
Image.asset('image name')

Steps to Add an Image:

1. Create a new folder

- It should be in the root of your flutter project. You can name it whatever you want, but *assets* are preferred.
- If you want to add other assets to your app, like fonts, it is preferred to make another subfolder named *images*.



2. Now you can copy your image to *images* sub-folder. The path should look like *images/yourImage*. Before adding images also check the above-mentioned supported image formats.
3. Register the assets folder in *pubspec.yaml* file and update it.

a) To add images, write the following code:

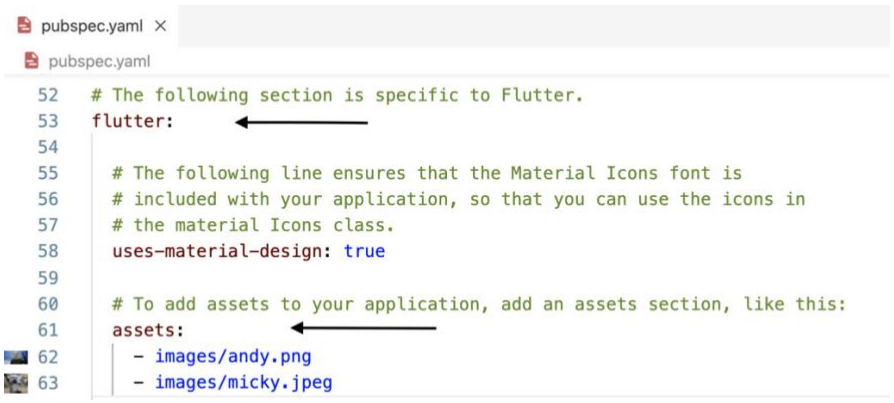
```
flutter:

  assets:

       - images/yourFirstImage.jpg

       - image/yourSecondImage.jpg
```

b) If you want to include all the images of the assets folder then add this:

```
flutter:
```

assets:

- images/

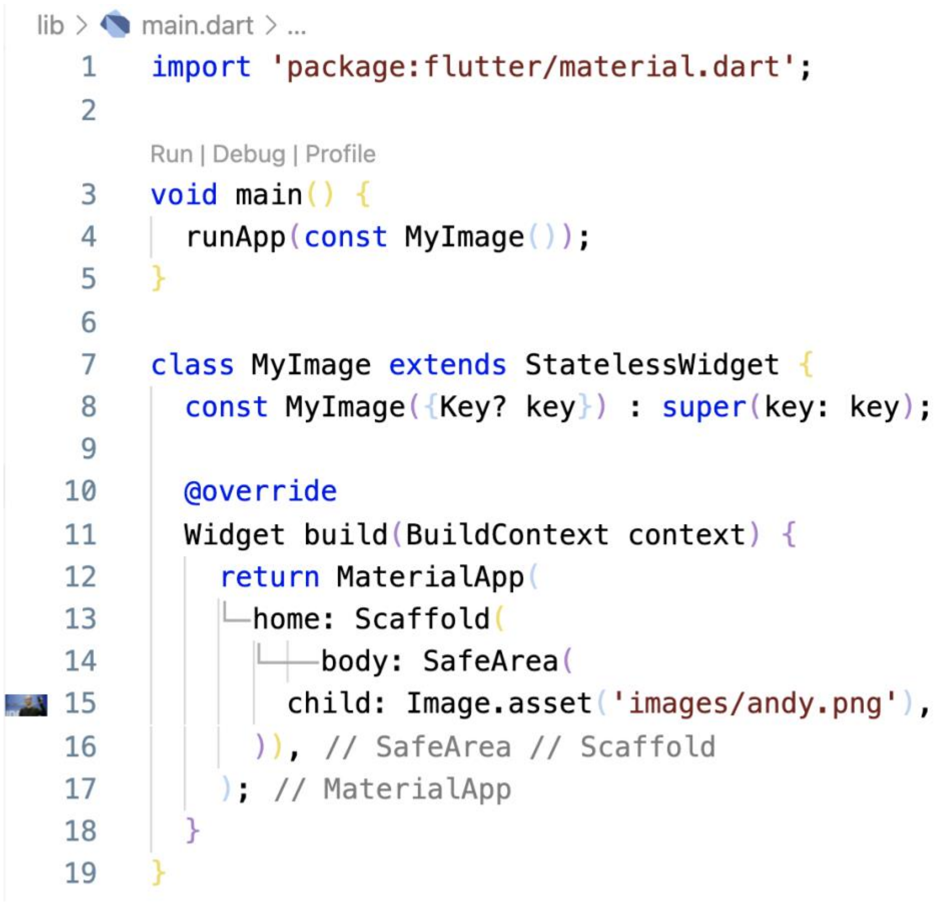


```
pubspec.yaml ×
pubspec.yaml
# The following section is specific to Flutter.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  assets:
    - images/andy.png
    - images/micky.jpeg
```

4. Insert the image code in the file, where you want to add the image.

Image.asset('images/andy.png')

Image. asset('images/micky.jpeg')



```
lib > main.dart > ...
import 'package:flutter/material.dart';

Run | Debug | Profile
void main() {
  runApp(const MyImage());
}

class MyImage extends StatelessWidget {
  const MyImage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: SafeArea(
          child: Image.asset('images/andy.png'),
        )), // SafeArea // Scaffold
    ); // MaterialApp
  }
}
```

❖ Image.network:

```
Image.network(
  'https://images.unsplash.com/photo-1547721064-da6cfb341d50',
  width: 280.0,
)
```

This is a simple code snippet showing the implementation of **Image.network** as an example. Here we are using 2 basic arguments that will be handy while using Image.network. The **url** and **width** of image are used in the example as arguments. The **width and height arguments** can be used to **control the size** of the image displayed. On execution the flutter will load the image from web or url given and display it inside the container.

Fit Argument for Image:

```
  Container(
                width: double.infinity,
                height: double.infinity,
                color: Colors.yellow,
                child: Image.network(
                  "https://images.unsplash.com/photo-1547721064-da6cfb341d50",
                  fit: BoxFit.cover,
                ),
              )
```

Instead of giving the width, you can fit the network image inside a container with the fit argument. Here we use Boxfit.contain which makes the image contained inside the Container as shown in the image. Values that can be given to the fit argument are:

- **BoxFit.contain** - As large as possible but contained within the Container
- **BoxFit.cover** - As small as possible but covering the entire Container

- **BoxFit.fill** - Fill the entire Container but may distort the image aspect ratio
- **BoxFit.fitHeight** - The height of the image will be equal to container. It will not mess with image aspect ratio
- **BoxFit.fitWidth** - The height of the image will be equal to container.
- **BoxFit.none** - Image is not resized at all
- **BoxFit.scaleDown** - Scales down the image to fit inside the Container

Image as the background of a Container:

```
 Container(
              alignment: Alignment.center,
              child: Container(
              width: 300.0,
              height: 500.0,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(30.0),
                image: DecorationImage(
                  image: NetworkImage(
                    "https://images.unsplash.com/photo-1547721064-da6cfb341d50",
                  ), fit: BoxFit.cover)
                ),
              ),
            ),
```

The code snippet above shows, how to display a network image as the background of a Container. For this, we use the **NetworkImage()** widget. Here **NetworkImage()** is given as the value for the image property of the **DecorationImage()** of a container. Fit argument can also be used for **DecorationImage()** to adjust the size of the network image shown.

❖ Image.file

To load images from the file system in the target device, you must use Image.file. However, you must first ensure that the app has the proper permissions to access the device's external storage. Open and edit your project's AndroidManifest.xml file to request these permissions:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package= "flutter.image.demo">
    <uses-permission android:name =
      "android.permission.READ_EXTERNAL_STORAGE"/>
  <application
  ...
  </application>
</manifest>
```

Find the image you wish to add and view its details to determine the image's file path.

```
import 'package:flutter/material.dart';
import 'dart:io';void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Flutter Image Demo',
   home: Scaffold(
    appBar: AppBar(
     title: const Text('Flutter Image Demo'),
    ),
    body: new Container(
     color: Colors.grey[200],
     child: Image.file(

new File('/storage/emulated/0/Download/forest.jpg')),
     alignment: Alignment.center,
    ),
   ),
  );
 }
}
```