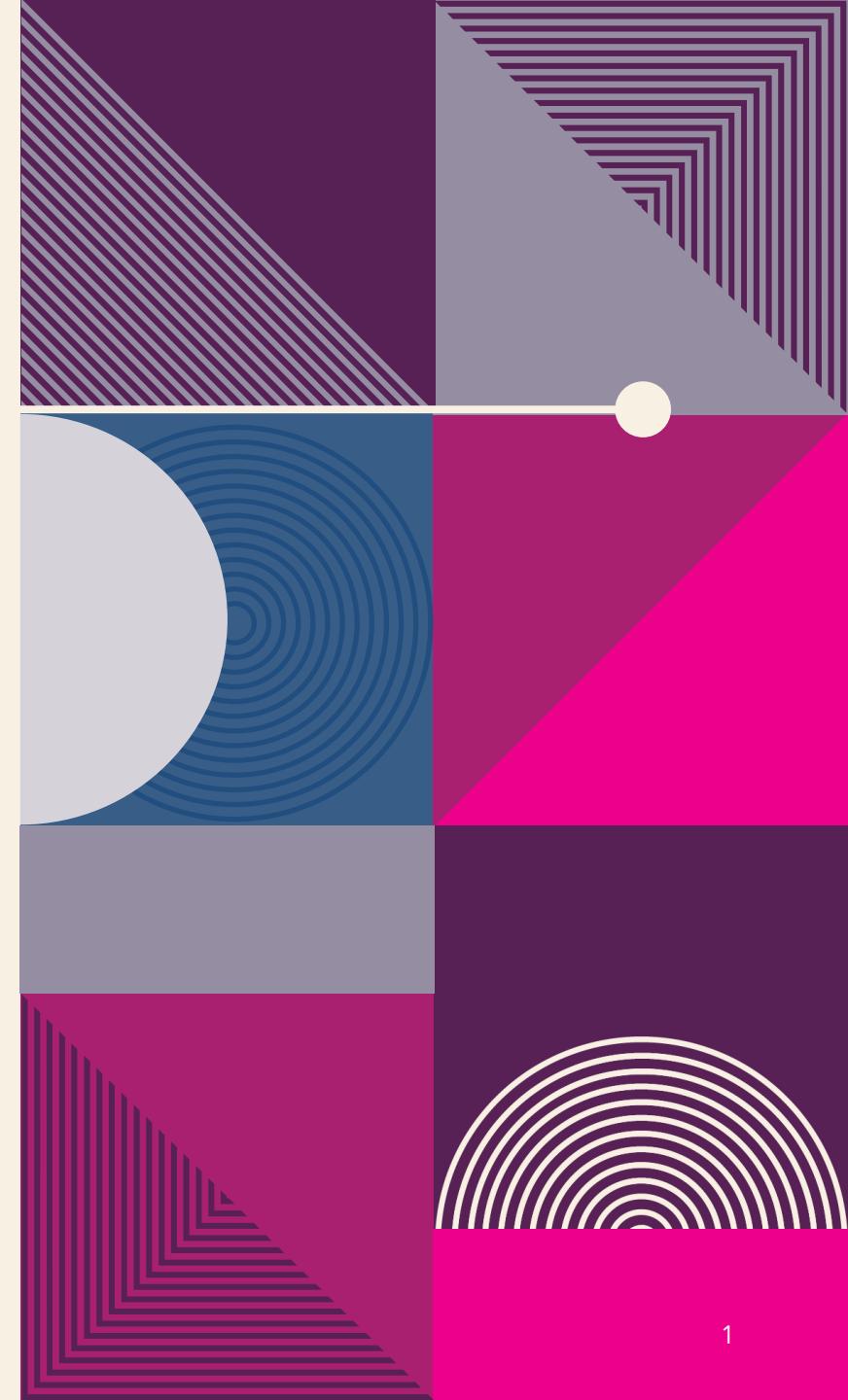


Using Common Widgets Images and Icons



USING IMAGES AND ICONS

AssetBundle

The **AssetBundle** class provides access to **custom resources** such as **images**, **fonts**, **audio**, **data files**, and more. Before a Flutter app can use a resource, you must **declare** it in the **pubspec.yaml** file.

1. declaring each asset (one by one)

```
// pubspec.yaml file to edit
# To add assets to your application, add an assets section, like this:
assets:
  - assets/images/logo.png
  - assets/images/work.png
  - assets/data/seed.json
```

2. declare all the assets in each directory

```
// pubspec.yaml file to edit
# To add assets to your application, add an assets section, like this:
assets:
  - assets/images/
  - assets/data/
```

Image

The **Image widget** displays an image from a **local** or **URL (web) source**.

To load an Image widget, there are a few different constructors to use.

- **Image()** : Retrieves image from an **ImageProvider** class
- **Image.asset()** : Retrieves image from an **AssetBundle** class using a **key**
- **Image.file()** : Retrieves image from a **File** class
- **Image.network()** : Retrieves image from a **URL** path

Icon

The **Icon** widget is drawn with a glyph from a font described in **IconData**. Flutter's **icons.dart** file has the full list of icons available from the font **MaterialIcons**.

Flutter Images

In this section, we are going to see how we can display images in Flutter. When you create an app in Flutter, it includes both code and assets (resources). An asset is a file, which is bundled and deployed with the app and is accessible at runtime. The asset can include static data, configuration files, icons, and images. The Flutter supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF, BMP, and WBMP.

Displaying images is the fundamental concept of most of the mobile apps. Flutter has an Image widget that allows displaying different types of images in the mobile application.

To display an image in Flutter, do the following steps:

Step 1: First, we need to create a new **folder** inside the root of the Flutter project and named it `images`. We can also give it any other name if you want.

Step 2: Next, inside this folder, add one image manually.

Step 3: Update the `pubspec.yaml` file. Suppose the image name is `tablet.jpeg`, then `pubspec.yaml` file is:

```
assets:  
  - images/tablet.jpeg
```

If the images folder contains more than one image, we can include it by specifying the directory name with the **slash (/)** character at the end.

flutter:

```
assets:  
  - images/
```

Step 4: Finally, open the **main.dart** file and insert the following code.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Image Demo'),
        ),
        body: Center(
          child: Column(
            children: <Widget>[
              Image.asset('images/tablet.jpeg'),
              Text(
                'A tablet is a wireless touch screen computer that is smaller than
a notebook but larger than a smartphone.',
                style: TextStyle(fontSize: 20.0),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```



A tablet is a wireless touch screen computer that is smaller than a notebook but larger than a smartphone.

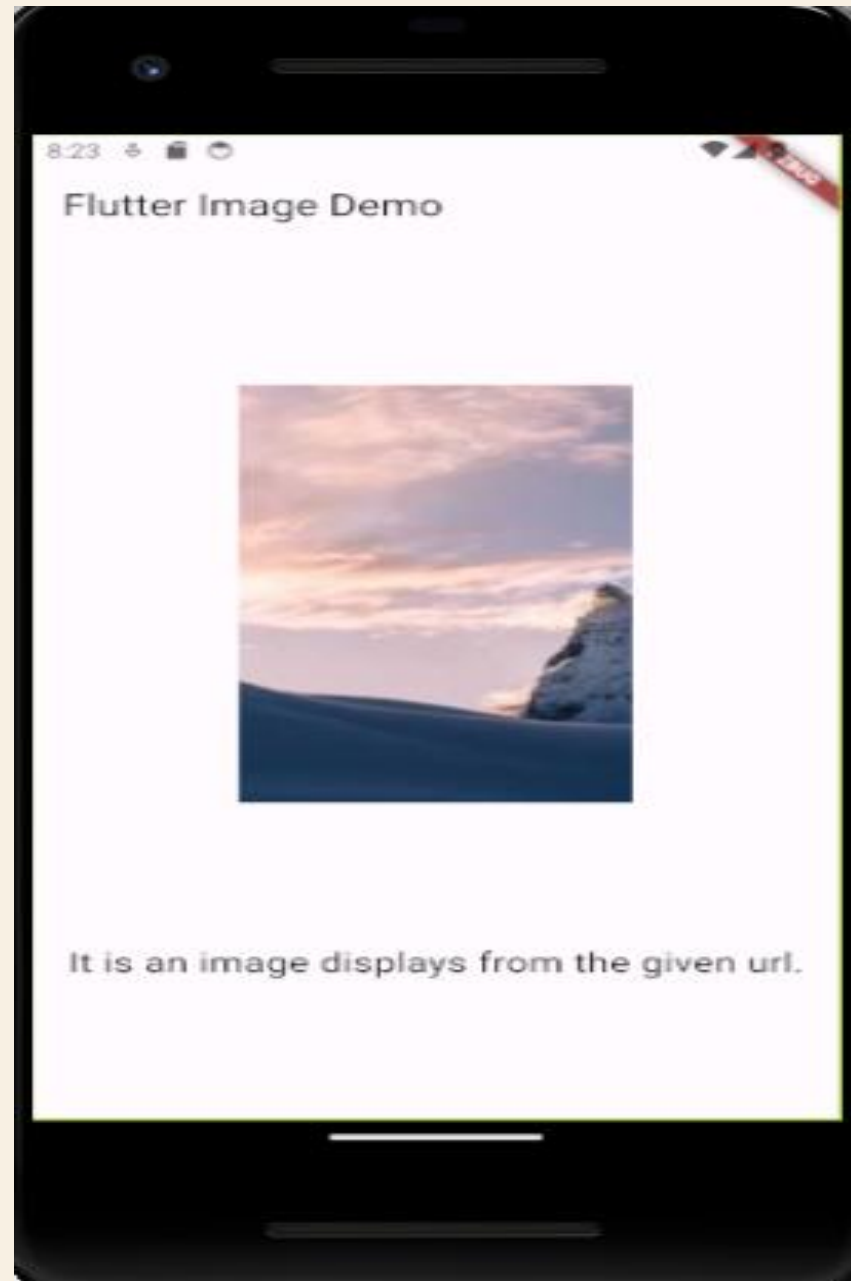
Display images from the internet

Displaying images from the internet or network is very simple. Flutter provides a built-in method **Image.network** to work with images from a URL. The Image.network method also allows you to use some optional properties, such as height, width, color, fit, and many more. We can use the following syntax to display an image from the internet.

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Image Demo'),
        ),
        body: Center(
          child: Column(
            children: <Widget>[
              Image.network('https://picsum.photos/seed/picsum/200/300',
                height: 400, width: 250),
              Text(
                'It is an image displays from the given url.',
                style: TextStyle(fontSize: 20.0),
              )
            ],
          ),
        ),
      ),
    );
  }
}
```



Creating the **Images** Project; Adding **Assets**; and **Loading Images, Icons, and Decorators**

Create a new Flutter project. For this project, you need to create only the **pages** and **assets/images** folders.

1. Open the **pubspec.yaml** file to add **resources**. In the **assets** section, add the **assets/images/** folder declaration.

```
# To add assets to your application, add an assets section, like this:  
assets:  
  - assets/images/
```

2. Open the **home.dart** file and **modify** the **body** property. Add a **SafeArea** widget to the **body** property with a **SingleChildScrollView** as a **child** of the **SafeArea** widget. **Add Padding** as a **child** of **SingleChildScrollView** and then add a **Column** as a **child** of the **Padding**.

```
body: SafeArea(  
  child: SingleChildScrollView(  
    child: Padding(  
      padding: EdgeInsets.all(16.0),  
      child: Column(  
        children: <Widget>[  
          ],  
        ),  
      ),  
    ),  
  ),  
)
```

2. Add the widget **class** name **ImagesAndIconWidget()** to the **Column** children widget list. The **Column** is located in the **body** property.

```
body: SafeArea(  
  child: SingleChildScrollView(  
    child: Padding(  
      padding: EdgeInsets.all(16.0),  
      child: Column(  
        children: <Widget>[  
          const ImagesAndIconWidget(),  
        ],  
      ),  
    ),  
  ),  
)
```

3. Add the `ImagesAndIconWidget()` widget class after `class Home extends StatelessWidget {...}`. In the widget class, a local image is loaded by the `AssetImage` class. Using the `Image.network` constructor an image is loaded by a URL string.

```
class ImagesAndIconWidget extends StatelessWidget {
  const ImagesAndIconWidget({
    Key key,
  }) : super(key: key);

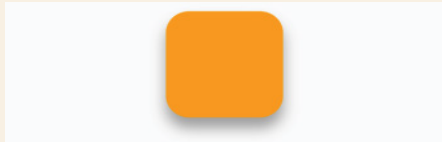
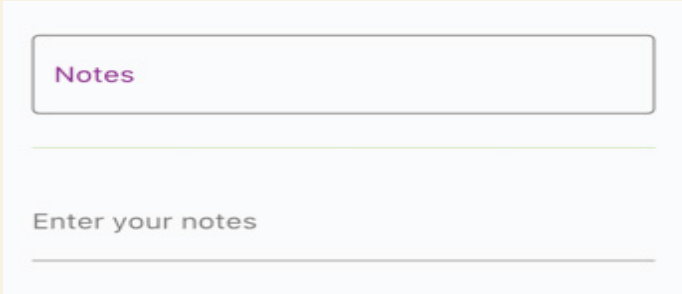
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: <Widget>[
        Image(
          image: AssetImage("assets/images/logo.png"),
          //color: Colors.orange,
          fit: BoxFit.cover,
          width: MediaQuery.of(context).size.width / 3,
        ),
        Image.network(
          'https://flutter.io/images/catalog-widget-placeholder.png',
          width: MediaQuery.of(context).size.width / 3,
        ),
        Icon(
          Icons.brush,
          color: Colors.lightBlue,
          size: 48.0,
        ),
      ],
    );
  }
}
```

USING DECORATORS

Decorators help to convey a message depending on the user’s action or customize the look and feel of a widget.

There are different types of decorators for each task.

- **Decoration**: The base class to **define other decorations**.
- **BoxDecoration**:. Provides many ways to draw a **box with border, body, and boxShadow**
- **InputDecoration**: Used in **TextField** and **TextFormField** to customize the **border, label, icon, and styles**.

BoxDecoration applied to a Container	InputDecoration with OutlineInputBorder and default border
	

Continuing the Images Project by Adding Decorators

Still editing the **home.dart** file

1. **Add** the widget **class** names **BoxDecoratorWidget()** and **InputDecoratorsWidget()** after the **ImagesAndIconWidget()** widget class. Add a **Divider()** widget between each widget class name.

```
body: SafeArea(  
  child: SingleChildScrollView(  
    child: Padding(  
      padding: EdgeInsets.all(16.0),  
      child: Column(  
        children: <Widget>[  
          const ImagesAndIconWidget(),  
          Divider(),  
          const BoxDecoratorWidget(),  
          Divider(),  
          const InputDecoratorsWidget(),  
        ],  
      ),  
    ),  
  ),  
)
```

2. Add the **BoxDecoratorWidget()** widget **class** after the **ImagesAndIconWidget()** widget **class**. The widget class returns a **Padding** widget with the **Container** widget as a **child**. The **Container decoration** property uses the **BoxDecoration** **class**. Using the **BoxDecoration** **borderRadius**, **color**, and **boxShadow** properties.

```
class BoxDecoratorWidget extends StatelessWidget {
  const BoxDecoratorWidget({
    Key key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(16.0),
      child: Container(
        height: 100.0,
        width: 100.0,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.all(Radius.circular(20.0)),
          color: Colors.orange,
          boxShadow: [
            BoxShadow(
              color: Colors.grey,
              blurRadius: 10.0,
              offset: Offset(0.0, 10.0),
            )
          ],
        ),
      ),
    );
  }
}
```

3. Add the `InputDecoratorsWidget()` widget class after the `BoxDecoratorWidget()` widget class. You take a `TextField` and use `TextStyle` to change the color and `fontSize` properties. The `InputDecoration` class is used to set the `labelText`, `labelStyle`, `border`, and `enabledBorder` values to customize the border properties.

```
class InputDecoratorsWidget extends StatelessWidget {
  const InputDecoratorsWidget({
    Key key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        TextField(
          keyboardType: TextInputType.text,
          style: TextStyle(
            color: Colors.grey.shade800,
            fontSize: 16.0,
          ),
          decoration: InputDecoration(
            labelText: "Notes",
            labelStyle: TextStyle(color: Colors.purple),
            //border: UnderlineInputBorder(),
            //enabledBorder: OutlineInputBorder(borderSide: BorderSide(color: Colors.lightGreen)),
            border: OutlineInputBorder(),
          ),
        ),
        Divider(
          color: Colors.lightGreen,
          height: 50.0,
        ),
        TextFormField(
          decoration: InputDecoration(labelText: 'Enter your notes'), ), ],
    );
  }
}
```

