15. Introduction to Widgets

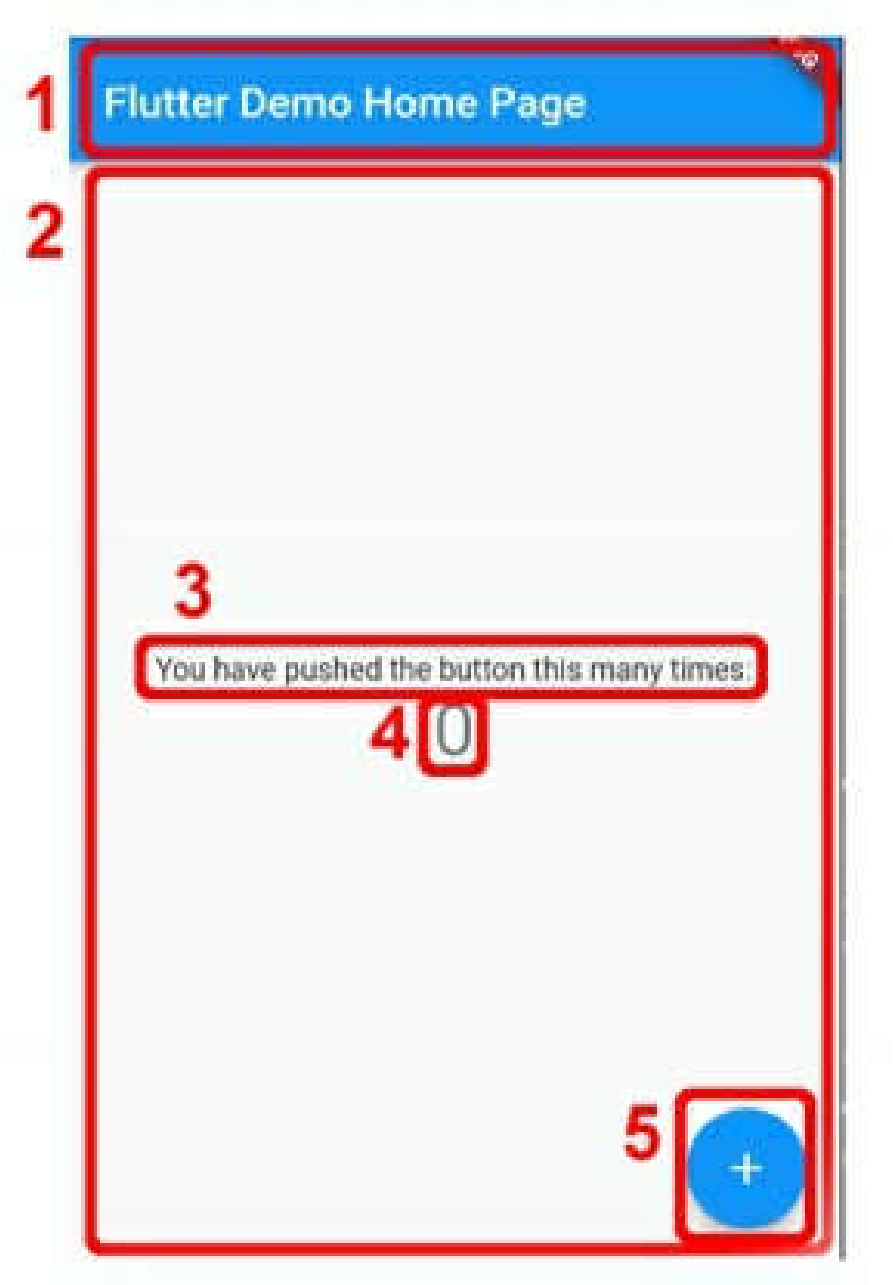
Introduction

The purpose of this chapter is to cover composition.

We mentioned composition earlier. It's how you compose your user interface from Widgets and each one is used to render a part of the UI. Widgets are built by composing other Widgets, which are themselves built out of progressively more basic Widgets. This is known as aggressive composability.

We also mentioned that your app ends up being a hierarchy of Widgets, a Widget Tree:

- Some widgets are parent widgets.
    - For example, Widget #2.
- , Some widgets are child widgets.
    - For example, Widget #3 and Widget #4 are children of Widget #2.



What Are Widgets?

Widgets are really configuration objects rather than graphic objects.
When you write a Widget, it is not just directly

rendered on screen, it’s not as direct as that.
You write them and they *configure the user interface* then Flutter gets them rendered on screen.

User Interface: Material & Cupertino

Google has its own user interface design language called Material, which is used in all Google products. If you look at a program running on an Android phone, chances are that the UI will have that look and feel.

Material Design, According to Google

Material Design is a system for building bold and beautiful digital products. By uniting style, branding, interaction, and motion under a consistent set of principles and components, product teams can realize their greatest design potential.

Most Flutter Widgets Work with Material Design

As Flutter was written by Google, most Flutter widgets support the Material design look and feel. As most people are writing Flutter apps that implement the Material design look and feel, we are going to concentrate on the Flutter widgets that support that

look and feel.

User Interface: Cupertino

Apple (based in Cupertono) is the other big player in mobile apps and it has its own user interface design language. Apple does not have a name for its design language (yet). Apple though has something called “Human Interface Guidelines”. These guidelines ensure that all iOS applications adhere to Apple’s design principles.

Flutter Includes iOS-Styled Widgets

Google has written many iOS-styled Widgets for Flutter developers so that they can emulate native iOS apps. I have not used these Widgets, so I am not going to spend any time on them. However, it is good to know that they exist and are available. Here is a list:

- CupertinoActionSheet
- CupertinoActivityIndicator
- CupertinoAlertDialog
- CupertinoButton
- CupertinoDatePicker
- CupertinoDialog
- CupertinoFullscreenDialogTransition
- CupertinoPageScaffold

- CupertinoPageTransition
- CupertinoPicker
- CupertinoPopupSurface
- CupertinoSegmentedControl
- CupertinoSlider
- CupertinoSwitch
- CupertinoNavigationBar
- CupertinoTabBar
- CupertinoTabScaffold
- CupertinoTabView
- CupertinoTextField
- CupertinoTimerPicker

Building Widgets

Flutter does the job of rendering the widgets on the screen for us (more on change detection & rendering later), but it needs *configuration information for the widget*: what color is it going to be, what is its border, does it contain other widgets....

Build Method

When it needs to know how to render a widget, Flutter calls the ‘build’ method in your widget. That method returns a Widget object that gives Flutter configuration information about the widget (and any child widgets

that it may be composed of).

```
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Center(child:Text('Hello World'))
    );
  }
}
```

The ‘build’ method takes one argument, the BuildContext (more on that later) and returns a Widget object. That returned Widget object contains configuration data that tells Flutter that it needs to render a Material App widget with a title and some centered text.

Build Context

Earlier we mentioned that a widget can contain other widgets, in a tree structure, a hierarchy. This is often called a Widget Tree.

The first argument to the build’ method of your Widget is the BuildContext. This gives your ‘build’ method

information about the location of your Widget in the Widget Tree.
It may not seem useful at the moment but will come in very handy later on!

Not All Widgets Are Equal

Ok, we know that a Flutter user interface is composed of Widgets and that each widget has a build method that gives Flutter information on how render it. That's true for all widgets.

We also know that some Widgets can be composed of other widgets, for example a Form widget being composed of text and input boxes.

However, in addition to that, some widgets are simple (stateless), others are more dynamic (stateful). These dynamic stateful Widgets can react to things happening, like data (state) changing.

State

Before we talk about state, we need to get our terminology straight.
State is simply information about something held in memory.

- In the context of an application, the state is the

data (information) it uses.

- In the context of a widget (part of the UI) the state is the data (information) contained in the widget'.

Mutable State

Mutable state is state that can be changed.

Immutable State

Immutable state is state that cannot be changed.

Flutter Widgets & State

Flutter has two types of Widgets: StatelessWidgets and StatefulWidgets.
Both types of Flutter Widgets can store data that doesn't change (immutable state).
Only one can appear to be able to store data that changes (mutable state)

Stateful Widgets

Their UI Can Update When State Changes

Stateful Widgets appear to be able to store data that changes (mutable state) and update their UI when that data changes.

For example, you would expect a stateful text input widget to store the data for the actual text displayed therein and that can change.

They Don’t Actually Store the State Themselves

Internally Stateful Widgets are only associated with a separate State object, which stores data that can change (mutable state). When the State needs to change (the user did something and an event occurred), the code invokes the ‘setState’ method in the State object and the Stateful Widget is re-rendered (the ‘build’ method is invoked to update the UI).

Stateless Widgets

Stateless Widgets can store data that doesn’t change.

- They are typically used to do simple things like ‘display customer name’.
- They are the most-commonly used Widgets.
- They don’t update their data.
- They don’t update their UI when data changes (their ‘build’ method isn’t invoked).
- They are very lightweight and are created and thrown away very often.

For example, if you have a Customer Name Stateless Widget and the customer name changes then a new Widget is created for the new name, it replaces the old one and the old one is thrown away.

Event Handling

Widgets have event handlers. These are essential so that the UI reacts to the actions of the user.

Event Handlers

For example, the IconButton below has an ‘onPressed’ event handler.

```
IconButton(
        icon: const Icon(Icons.rotate_right),
        tooltip: 'Brightness',
        onPressed: () {
          setState(() {
            ModelBinding.update(context, new
ThemeModel(!model.dark));
          });
        },
```

```
        )
```

When you need to respond to an event, you write a function and assign it to the event handler:

```
onPressed: () {
          setState(() {
            ModelBinding.update(context, new
ThemeModel(!model.dark));
          });
        }
```

Styling

Margins, Borders & Padding

Most Widgets enable you to specify margins and padding. Some also let you specify the border (for example Container). You will find these very useful when you want to get your UI to look exactly right.

- The padding is the spacing between the border and the inside of the Widget.
- The border is the border around the outside of the Widget.
- The margin is the spacing around the outside of the Widget and other Widgets.

Further Reading

https://medium.com/fluttery/what-even-are-flutter-widgets-ce537a048a7d
https://medium.com/flutter-io/why-flutter-doesnt-use-oem-widgets-94746e812510

16. Stateless Widgets

Introduction

The purpose of this chapter is to introduce stateless widgets and how they can be used.

Not All Widgets Need to be Smart

If you look a user interface, it consists of many Widgets but not many of them have to be smart or interact with the user.

If you look at the default flutter application, there are several widgets but only in fact one Widget with any interactions with the user – the 'MyHomePage' Widget that has a counter that counts up when the user clicks on the floating button.

So, the rest of the widgets are used to display something, not interact with the user. That is what stateless widgets are for.

Minimum Code

Here is the minimum code you need for a Stateless Widget:

```
class EmptyWidget extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) {
    return <Insert Some Widgets Here>;
  }
}
```

Creation

Stateless widgets are created by a parent widget in its ‘build’ method. They are given the information they need to do their job when they are created.

Stateless widgets receive arguments (information) from their parent widget in the ‘build’ method, which they store in final member variables.

Example

```
CarWidget("Bmw", "M3",
          "https://media.ed.edmunds-
media.com/bmw/m3/2018/oem/2018_bmw_m3_sedan_base_fq_o
```

- ‘Bmw’
    - Stored in member variable ‘make’.
- ‘M3’
    - Stored in member variable ‘model’.
- "https://media.ed.edmunds-media.com/bmw/m3/2018/oem/2018_bmw_m3_s
    - Stored in member variable ‘imageSrc’.

Rendering

The ‘Build’ Method

- Stateless Widgets generate their UI in their ‘build’ method, the result of which is rendered by Flutter.
- They can build their UI using values from their member variables, or from other sources.
- They cannot force themselves to re-render.

Values from Member Variables

When a Stateless Widget is asked to build a UI, it can use the values from these member variables to render the UI (probably with other Stateless Widget children). These values **don’t** change, they are set in the constructor and that’s it.

Example

The code below builds a UI to display textual info about a car (its make & model) using information from the member variables.

```
@override
  Widget build(BuildContext context) {
    return Center(
        child: Column(children: <Widget>[
```

```
      Text(make),
      Text(model),
      Image.network(imageSrc)
    ]));
  }
```

Values from Other Sources

When a Stateless Widget is asked to build a UI, it can use values from other sources, for example InheritedWidgets (which can store information).

Example

The code below builds a UI to say “Hi There”, using information from another source (the ‘Theme’ inherited widget) to determine text color.

```
@override
Widget build(BuildContext context) {
return Center(
    child: Column(children: <Widget>[
    Text("Hello", style:
Theme.of(context).textTheme.display1),
    Text("There", style: Theme.of(context).textTheme.display1)
]));
}
```

When Does The ‘Build’ Method

Execute?

- The first time the widget is inserted in the tree.
- When the widget's parent changes.
- When the values in another source change, for example when an InheritedWidget it depends on changes.

Lifecycle

These widgets are throw-away widgets, they don’t hang around.
You create them in the ‘build’ method of another widget, and they are re-created every time that ‘build’ of the parent widget runs.

Exercise – ‘first_stateless’

We start off by creating a basic app with Stateless Widgets.
Later on, we enhance it to make it look more attractive.

Step 1 – Create Default Flutter App

Follow the instructions in Generate Your First App
Leave the project open.

Step 2 – Replace Application Code

Replace contents of file ‘main.dart’ in folder ‘lib’ with the following:

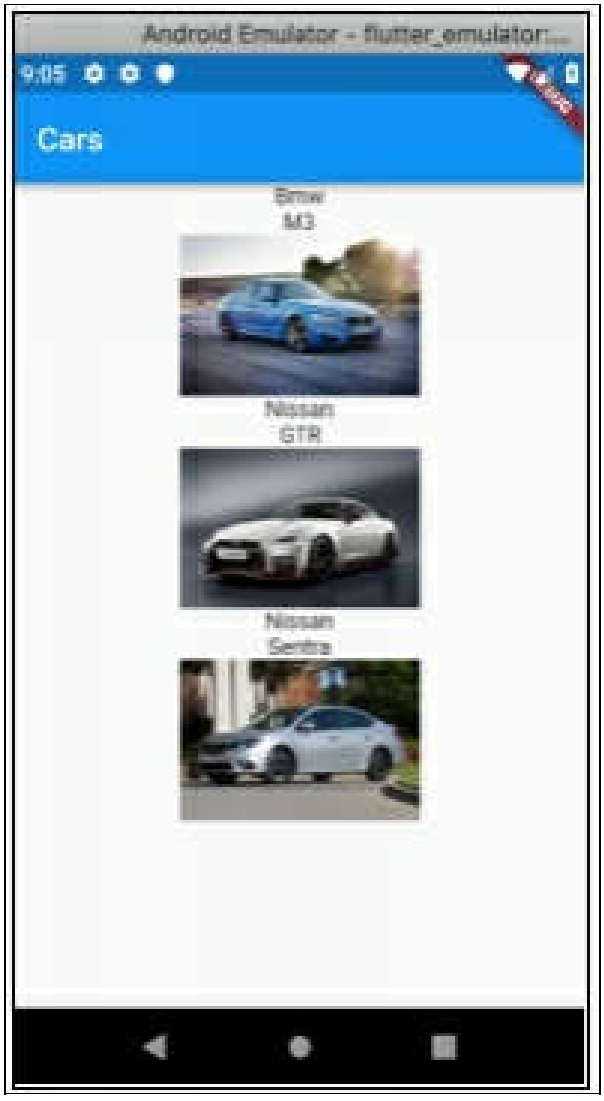
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new MyHomePage(title: 'Cars'),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(
```

```
        title: new Text(this.title),
      ),
      body: new Column(children: <Widget>[
        CarWidget("Bmw", "M3",
            "https://media.ed.edmunds-
media.com/bmw/m3/2018/oem/2018_bmw_m3_sedan_base_fq_o
        CarWidget("Nissan", "GTR",
            "https://media.ed.edmunds-media.com/nissan/gt-
r/2018/oem/2018_nissan_gt-
r_coupe_nismo_fq_oem_1_150.jpg"),
        CarWidget("Nissan", "Sentra",
            "https://media.ed.edmunds-
media.com/nissan/sentra/2017/oem/2017_nissan_sentra_sedan_s
turbo_fq_oem_4_150.jpg"),
      ]));
  }
}

class CarWidget extends StatelessWidget {
  CarWidget(this.make, this.model, this.imageSrc) : super();

  final String make;
  final String model;
  final String imageSrc;

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Column(children: <Widget>[
```

```
      Text(make),
      Text(model),
      Image.network(imageSrc)
    ]));
  }
}
```

Step 3 – Open Emulator & Run

Follow the instructions in ‘Open Emulator & Run Your First App’ to run the app.
You should get something like the following:



Summary So Far

- The MyApp & Material App Widgets are unchanged.
- The MyHomePage Widget is unchanged except for the build method, which now contains a Column Widget (see below) containing 3 Car Widgets. Note how we pass the information to

each Car Widget in the constructor.

- We have a new StatelessWidget called CarWidget. It accepts data in the constructor. In the build method it returns a Center Widget (see below) that contains a Column Widget (see below) that contains 3 widgets: a Text Widget for the make, another for the model and an Image Widget for the image.
- Widgets used (more info about widgets in Chapter ‘Flutter Widgets’).
    - Column Widget
        - Layout Widget that displays its children vertically.
    - Center Widget
        - Layout Widget that centers its child.
    - Text Widget
        - Displays text.
    - Image Widget
        - Displays an image.

Step 4 – Add Some Padding

Now let’s add some more vertical padding between each car to spread them out a bit. This is achieved by wrapping the existing Center Widget in the ‘build’

method in the CarWidget with a Padding Widget. Note how the Padding constructor requires a ‘padding’ argument and a ‘child’ argument.

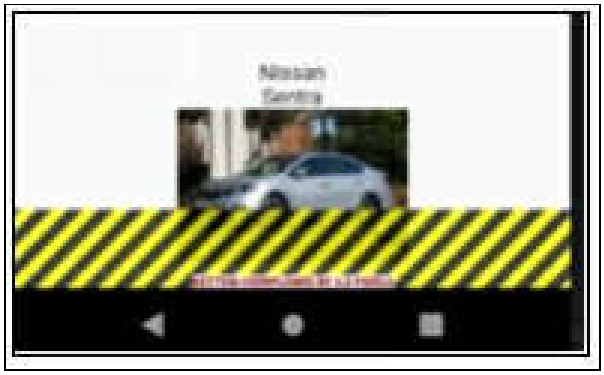
Change the ‘build’ method in the CarWidget to the following:

```
@override
Widget build(BuildContext context) {
  return Padding(
      padding: EdgeInsets.all(20.0),
      child: Center(
        child: Column(children: <Widget>[
        Text(make),
        Text(model),
        Image.network(imageSrc)
      ]))));
}
```

Now the cars are more spaced out.

Step 5 – Add Scrolling

Depending on how your emulator is setup, you may see Chevrons at the bottom. This is because you have run out of vertical space.



The remedy for this is simple. Edit the MyHomePage Widget and change the Column (the one that contains the CarWidgets) to a ListView.

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text(this.title),
    ),
    body: new ListView(children: <Widget>[
      CarWidget(“Bmw”, “M3",
          “https://media.ed.edmunds-
media.com/bmw/m3/2018/oem/2018_bmw_m3_sedan_base_fq_o
      CarWidget(“Nissan”, “GTR”,
          “https://media.ed.edmunds-media.com/nissan/gt-
r/2018/oem/2018_nissan_gt-
r_coupe_nismo_fq_oem_1_150.jpg”),
      CarWidget(“Nissan”, “Sentra”,
          “https://media.ed.edmunds-
media.com/nissan/sentra/2017/oem/2017_nissan_sentra_sedan_s
turbo_fq_oem_4_150.jpg”),
```

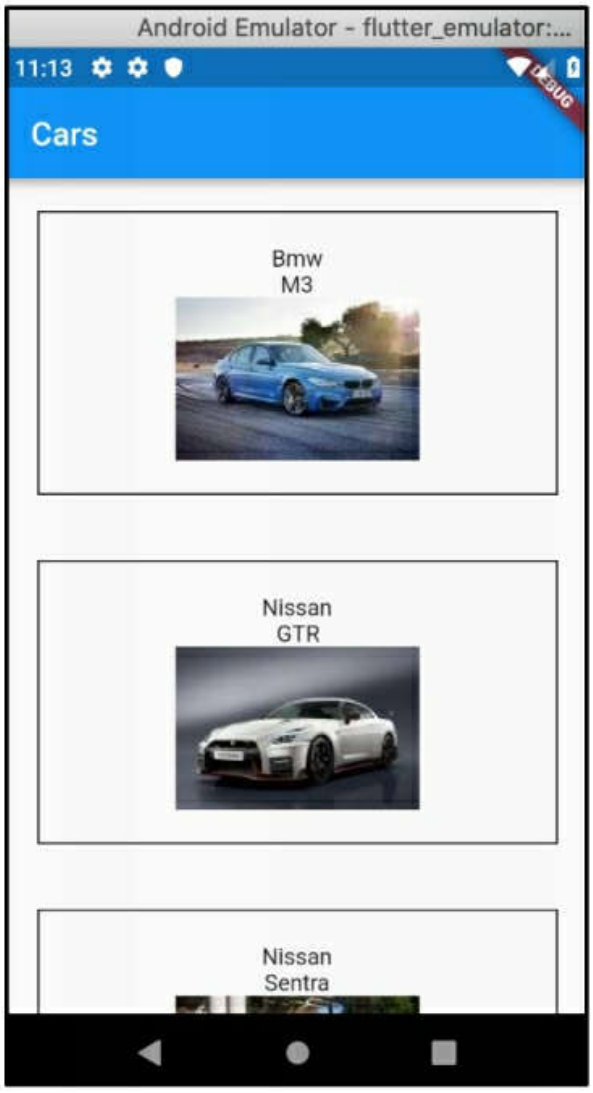
```
      ]));
}
```

Step 6 – Add Border

Let’s add a border around each car. This is achieved by wrapping the existing Center Widget in the ‘build’ method in the CarWidget with a Container Widget which has a border decoration and padding.

```
@override
Widget build(BuildContext context) {
  return Padding(
      padding: EdgeInsets.all(20.0),
      child: Container(
       decoration: BoxDecoration(border: Border.all()),
       padding: EdgeInsets.all(20.0),
       child: Center(
            child: Column(children: <Widget>[
          Text(make),
          Text(model),
          Image.network(imageSrc)
        ]))));
}
```

Looks much nicer now:



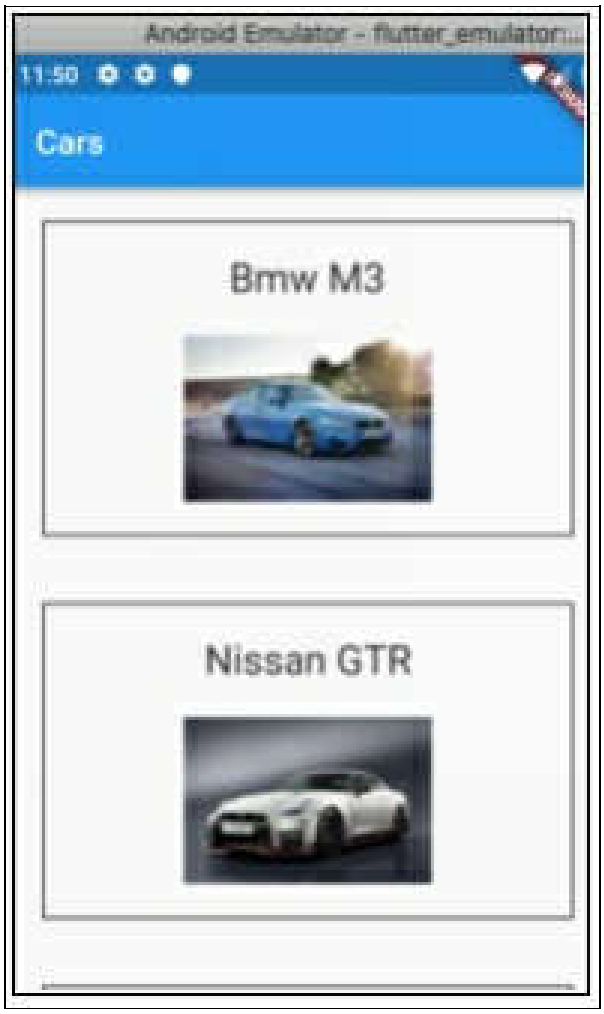
Step 7 – Final Touch

As a final touch, lets:

- Combine the make and model together using string interpolation.
- Change the make and model text style to be bigger.

- Add some padding between text and image.
  - Wrap image with padding at top.

```
@override
Widget build(BuildContext context) {
  return Padding(
    padding: EdgeInsets.all(20.0),
    child: Container(
      decoration: BoxDecoration(border: Border.all()),
      padding: EdgeInsets.all(20.0),
      child: Center(
        child: Column(children: <Widget>[
          Text('${make} ${model}', style:
TextStyle(fontSize: 24.0)),
          Padding(
            padding: EdgeInsets.only(top: 20.0),
            child: Image.network(imageSrc))
        ])));
}
```



Example – ‘stateless_widget_rebuild’

Optional

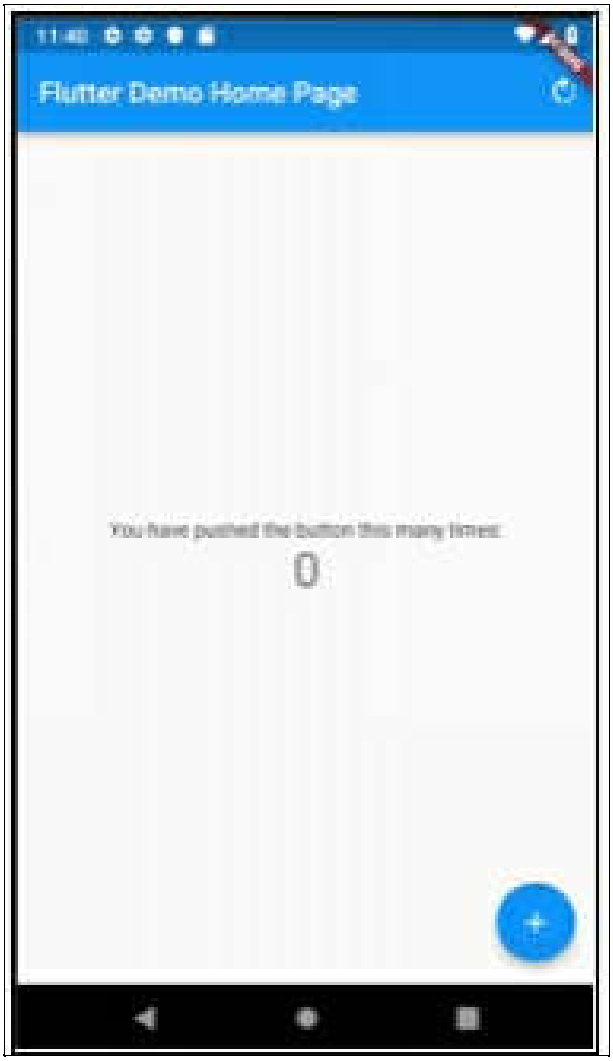
You don’t have to look at this example code at this point as it can get complicated. You might want to come back to this later once you want to look into StatelessWidgets in more detail. So, feel free to skip this and go onto the next chapter.

Purpose

I wrote this example to validate some of what I had written in this chapter was correct. I wanted to prove that a Stateless Widget can be re-rendered (in the 'build' method) when a parent Widget changes, without it being thrown away and reconstructed. This is contrary to some information I had read online that says that Stateless Widgets are only built once.

In this example, this is proven by the 'MyApp' Stateless Widget. I added console logging to the constructor and the 'build' method to see when it is constructed and re-rendered (in the 'build' method).

This app is similar to the default Flutter App except it has a button on the toolbar (top right) to change the theme's brightness. When you hit the toolbar, it updates a model which is part of a parent Widget higher up in the Widget Tree. This doesn't force the 'MyApp' Stateless Widget to be reconstructed but it does force it to invoke the 'build' method to rebuild the UI darker or lighter, as you can see from the Console Output below.


Flutter Demo Home Page
You have pushed the button this many times:
0

Console Output

Startup

```
I/flutter ( 5858): MyApp - constructor
I/flutter ( 5858): MyApp – build
```

Hit Button on Toolbar

```
I/flutter ( 5858): MyApp - build
```

Source Code

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

void main() => runApp(ModelBinding<ThemeModel>(
    initialModel: ThemeModel(true), child: new MyApp()));

class ThemeModel {
  ThemeModel(this._dark);

  bool _dark = true;

  bool get dark => _dark;

  @override
  bool operator ==(Object other) {
    if (identical(this, other)) {
      return true;
    } else if (other.runtimeType != runtimeType) {
      return false;
    } else {
      final ThemeModel otherModel = other;
      return dark == otherModel.dark;
    }
  }

  int get hashCode => dark.hashCode;
}

class _ModelBindingScope<T> extends InheritedWidget {
  const _ModelBindingScope({Key key,
```

```
this.modelBindingState, Widget child})
      : super(key: key, child: child);

  final _ModelBindingState<T> modelBindingState;

  @override
  bool updateShouldNotify(_ModelBindingScope oldWidget)
=> true;
}

class ModelBinding<T> extends StatefulWidget {
  ModelBinding({Key key, @required this.initialModel,
this.child})
      : assert(initialModel != null),
        super(key: key);

  final T initialModel;
  final Widget child;

  _ModelBindingState<T> createState() =>
_ModelBindingState<T>();

  static Type _typeOf<T>() => T;

  static T of<T>(BuildContext context) {
    final Type scopeType = _typeOf<_ModelBindingScope<T>>
();
    final _ModelBindingScope<T> scope =
        context.inheritFromWidgetOfExactType(scopeType);
    return scope.modelBindingState.currentModel;
```

```
  }

  static void update<T>(BuildContext context, T newModel) {
    final Type scopeType = _typeOf<_ModelBindingScope<T>>
();
    final _ModelBindingScope<dynamic> scope =
        context.inheritFromWidgetOfExactType(scopeType);
    scope.modelBindingState.updateModel(newModel);
  }
}

class _ModelBindingState<T> extends
State<ModelBinding<T>> {
  T currentModel;

  @override
  void initState() {
    super.initState();
    currentModel = widget.initialModel;
  }

  void updateModel(T newModel) {
    if (newModel != currentModel) {
      setState(() {
        currentModel = newModel;
      });
    }
  }
```

```
  @override
  Widget build(BuildContext context) {
    return _ModelBindingScope<T>(
      modelBindingState: this,
      child: widget.child,
    );
  }
}

class MyApp extends StatelessWidget {
  MyApp() {
    debugPrint('MyApp - constructor');
  }

  @override
  Widget build(BuildContext context) {
    debugPrint('MyApp - build');
    ThemeModel model = ModelBinding.of(context);
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
          primarySwatch: Colors.blue,
          brightness: model.dark ? Brightness.dark :
Brightness.light),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
```

```
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    ThemeModel model = ModelBinding.of(context);
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
        actions: <Widget>[
          IconButton(
            icon: const Icon(Icons.rotate_right),
            tooltip: 'Brightness',
```

```
          onPressed: () {
            setState(() {
              ModelBinding.update(context, new
ThemeModel(!model.dark));
            });
          },
        )
      ],
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.display1,
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
```

```
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

17. Stateful Widgets

Introduction

The purpose of this chapter is to introduce stateful widgets and how they can be used.

Some Widgets Need to be Smart

Stateful widgets are useful when the part of the user interface you are describing can change dynamically. User interfaces need to respond to a variety of things:

- The user doing something in the user interface.
- Receiving data from another computer.
- Time passing.

This is what Stateful Widgets are for. They store data (state) in an associated State class and they can respond when that data (state) changes as the result of the user doing something.

Minimum Code

Here is the minimum code you need for a Stateful Widget:

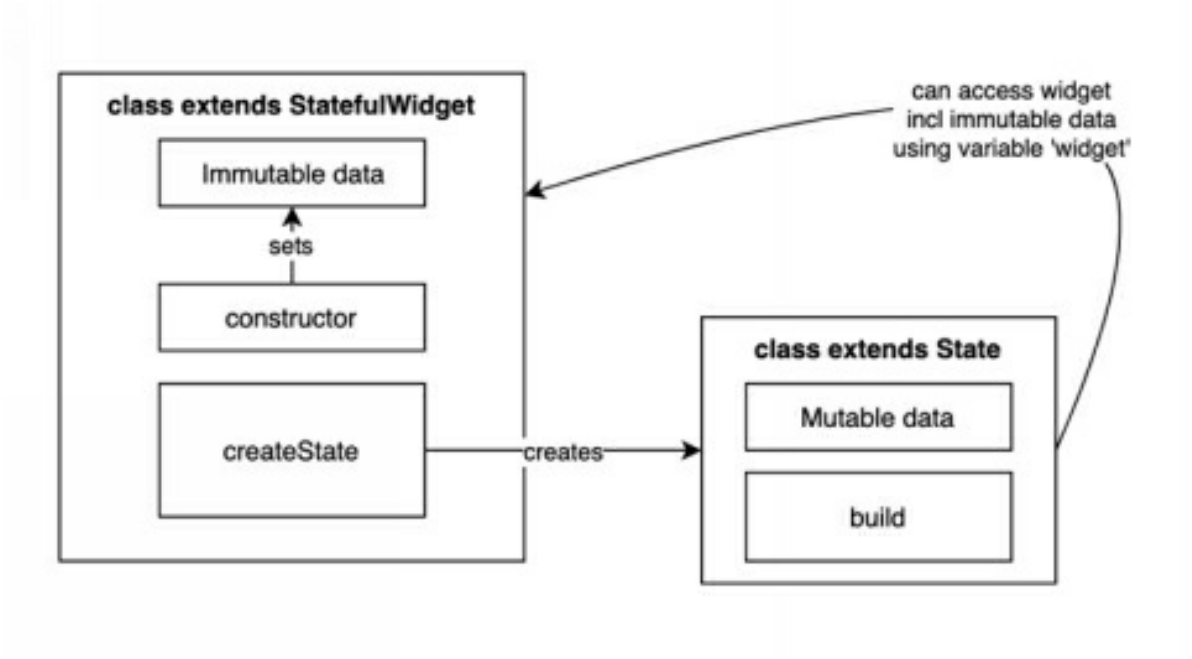
```
class EmptyWidget extends StatefulWidget {
  EmptyWidget({Key key}) : super(key: key);
```

```
  @override
  _EmptyWidgetState createState() => _EmptyWidgetState();
}

class _EmptyWidgetState extends State<EmptyWidget> {

  @override
  Widget build(BuildContext context) {
    return <Insert Some Widgets Here>;
  }
}
```

Two Classes

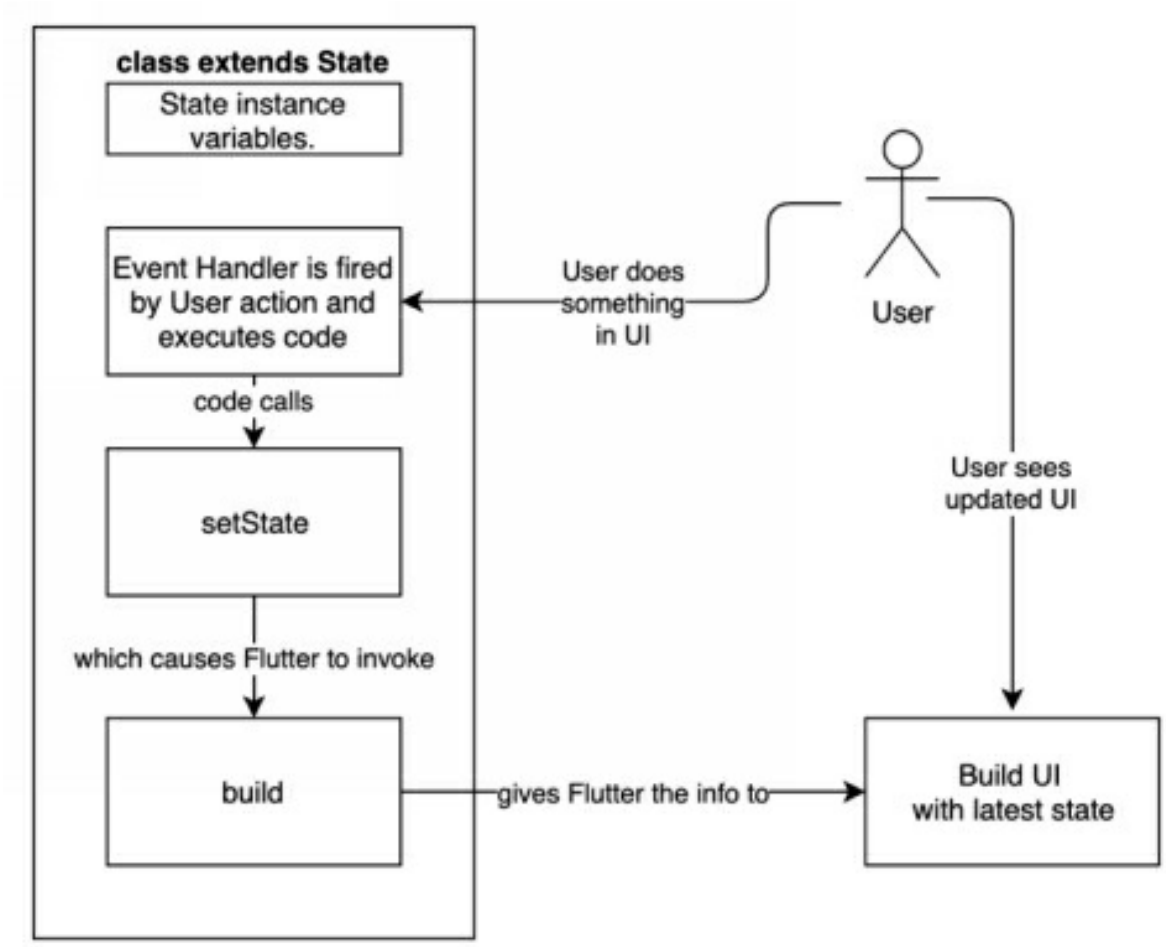
If you look at the minimum code above you will see that a Stateful Widget is composed of two classes, not one. You have one class that extends StatefulWidget, another that extends State.

Class #1 – the class that extends StatefulWidget



- This is a class that is used to create the State object, class #2 in its ‘createState’ method.
- An instance of this class is shorter-lived than that for the State object, class #2
- The data in this class cannot change (immutable).
    - It is final and passed in through the constructor, same as for a StatelessWidget.
    - This class is thrown away and replaced when the data needs to change, and a new Widget is constructed.

Class #2 – the class that extends State



- This is the class that does most of the work.
    - It holds the data that can change (mutable).
    - It builds the UI using the ‘build’ method.
    - It can respond to events, like the user clicking on a button.
- An instance of this class is longer-lived than that for the StatefulWidget, class #1.
- The data in this class can change.
    - Change the data within a lambda within

the ‘setState’ method and this will ensure the UI is rebuilt.
    - The StatefulWidget class #1 can be thrown away and replaced and this state is then attached to the replacement.
- Code in this class can refer to class #1 using the ‘widget’ variable.
- Sometimes this class uses an underscore prefix in its name, instance variables or methods.
    - In Dart the underscore prefix specifies something as private and cannot be used outside the dart file in which it is declared.
    - This protects this class, its instance variable or methods being used in another part of the project (not a good idea).

Creation

When you create a Stateful Widget the following happens.

1. The instance of class #1 (the class that extends StatefulWidget) is constructed.
2. The lifecycle method ‘createState’ of class #1 (the class that extends StatefulWidget) is invoked by Flutter to create the instance of class #2 (the

class that extends State).

3. The instance of class #2 (the class that extends State) is constructed.
4. The method ‘build’ of the State class (created in 3) is invoked to build the UI.

Rendering

The ‘Build’ Method

- Stateful Widgets generate their UI in their ‘build’ method, the result of which is rendered by Flutter.
    - That ‘build’ method resides in class #2, the class that extends State.
- They can build their UI using values from their member variables, other sources.
- They **can** force themselves to re-render.
- When the Stateful Widget method ‘setState’ is called in the State class, this invokes regeneration of the UI because it causes Flutter to invoke the ‘build’ method.
    - If you look at the default Flutter application, you will see this method to increment the counter. Note how it updates the instance variable ‘_counter’ in

a lambda inside the ‘setState’ method. This ensures that the UI will be rebuilt with the new counter value.

```
void _incrementCounter() {
   setState(() {
      _counter++;
   });
}
```

LifeCycle Methods

Class #1 – the class that extends StatefulWidget

createState()

Flutter calls this method. You add code here to an instance of the State class (class #2).

Class #2 – the class that extends State

build()

Flutter calls this method when the Widget has to be re-rendered (rebuilt).

initState()

Flutter calls this method when the widget is created,

after the constructor. This is a great place to add animation code to setup your AnimationController. See the Animation chapter for more info.

Example

```
@override
void initState() {
  super.initState();
  animation = new AnimationController(
    vsync: this,
    duration: new Duration(seconds: 3),
  );
  animation.addListener(() {
    this.setState(() {});
  });
}
```

didChangeDependencies ()

Flutter calls this method when Flutter detects that the data from another source has changed, possibly affecting the UI and causing a call to ‘build’. This could be caused by some data changing in an InheritedWidget higher up in the Widget tree. This not fired when ‘setState()’ is fired to rebuild the UI.

didUpdateWidget()

Flutter calls this method when it has to throw away the

StatefulWidget (class #1) and replace it with another StatefulWidget (class #1) of the same type but with different data, which is then associated with State (class #2). Now that the State is associated with a different StatefulWidget.

setState()

You call this method to set state in the Widget and ensure it rebuilds the UI using the ‘build’ method.

deactivate()

Rarely used. Flutter calls this method when State is removed from the tree, but it might be reinserted before the current frame change is finished. This method exists basically because State objects can be moved from one point in a tree to another.

dispose()

Flutter calls this method when 'dispose()' is called when the State object is destroyed. This is a great place to add code to dispose of instance variables, such as AnimationControllers:

```
@override
void dispose() {
  animation.dispose();
  super.dispose();
}
```

More Reading

https://www.didierboelens.com/2018/06/widget---state---context---inheritedwidget/

Example – ‘stateful_widget_flowers’

Optional

You don’t have to look at this example code at this point as it can get complicated. You might want to come back to this later once you want to look into StatefulWidgets in more detail. So, feel free to skip this and go onto the next chapter.

App Purpose

This app allows the user to view flowers then blur them.
There is a button on the top right which switches between ‘dark’ mode and ‘bright’ mode.


Flower


Flower


Flower

The point of the app is not to view flowers but to let you:

- See the logs when you run the app, so that you can figure out how Stateful Widgets work and when their lifecycle methods are fired.
- Run the code yourself, put breakpoints in and figure out how Stateful Widgets work with their

lifecycle events.

App Widgets

This app has two StatefulWidgets:

- AppWidget

  It is the main app and it contains the FlowerWidget.
  It has a state object _AppWidgetState that stores brightness.

- FlowerWidget

  It displays the flower in a frame with a title bar, a toolbar and an action button.
  It has a state object _FlowerWidgetState that stores the amount of blurring of the flower.

Start App

When you start the app, you will see the following logs:

```
I/flutter (23225): AppWidget - constructor - 261774211
I/flutter (23225): AppWidget - createState - 261774211
I/flutter (23225): _AppWidgetState - build - 160341789
I/flutter (23225): FlowerWidget - constructor - 1026133623
I/flutter (23225): FlowerWidget - createState - 1026133623
```

```
I/flutter (23225): _FlowerWidgetState - constructor - 514586671
I/flutter (23225): _FlowerWidgetState - initState - 514586671
I/flutter (23225): _FlowerWidgetState - didChangeDependencies - 514586671
I/flutter (23225): _FlowerWidgetState - build – 514586671
```

As you can see it creates each Widget first then creates the state using the ‘createState’ method. Note that ‘didChangeDependencies’ was invoked because the Theme state was set when the _AppWidgetState was built for the first time. The Theme is an InheritedWidget and used by the _FlowerWidgetState when it builds the UI.

Change the Flower

When you change the flower (on the toolbar), you will see the following UI change occur:

changes
to ->

with the following logs:

```
I/flutter (23700): _AppWidgetState - build - 543277124
I/flutter (23700): FlowerWidget - constructor - 814857920
I/flutter (23700): _FlowerWidgetState - didUpdateWidget - 57066142
I/flutter (23700): _FlowerWidgetState - build - 57066142
I/flutter (23700): _FlowerWidgetState - didChangeDependencies - 57066142
I/flutter (23700): _FlowerWidgetState - build - 57066142
I/flutter (23700): _FlowerWidgetState - didChangeDependencies - 57066142
I/flutter (23700): _FlowerWidgetState - build – 57066142
```

Note that this changes the ‘bright’ state of the AppWidget. This causes the AppWidget UI to be rebuilt with a different theme brightness and a different flower image. The ‘build’ method in the

_AppWidgetState creates a new FlowerWidget, because its constructor value ‘imageSrc’ has changed. That results in Flutter invoking the 'didUpdateWidget’ to indicate that the State is now associated with a different StatefulWidget. Flutter also invokes ‘didChangeDependencies’ because the Flower object is dependent on the Theme InheritedWidget and that was changed (the theme brightness was changed).

Add Blur

When hit the floating button at the bottom, you will see the flower image blur and you will see the following log:

```
I/flutter (23700): _FlowerWidgetState - build – 57066142
```

Note that Flutter invokes the ‘build’ in the FlowerWidget State object (see the code below) because the code calls the ‘setState’ method. No other lifecycle methods are invoked because that code does not affect another other widgets.

```
void _blurMore() {
  setState(() {
    _blur += 5.0;
  });
}
```

Source Code

```
import 'dart:ui';

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

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

class AppWidget extends StatefulWidget {
  AppWidget() {
    debugPrint("AppWidget - constructor - " +
hashCode.toString());
  }

  @override
  _AppWidgetState createState() {
    debugPrint("AppWidget - createState - " +
hashCode.toString());
    return _AppWidgetState();
  }
}

class _AppWidgetState extends State<AppWidget> {
  bool _bright = false;

  _brightnessCallback() {
    setState(() => _bright = !_bright);
  }
```

```
  @override
  Widget build(BuildContext context) {
    debugPrint("_AppWidgetState - build - " +
hashCode.toString());
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
            primarySwatch: Colors.blue,
            brightness: _bright ? Brightness.light :
Brightness.dark),
        home: FlowerWidget(
            imageSrc: _bright
                ? "https://www.viewbug.com/media/mediafiles/" +
                    "2015/07/05/56234977_large1300.jpg"
                : "https://images.unsplash.com/" +
                    "photo-1531603071569-0dd65ad72d53?ixlib=rb-
1.2.1&ixid=" +
                    "eyJhcHBfaWQiOjEyMDd9&w=1000&q=80",
            brightnessCallback: _brightnessCallback));
  }
}

class FlowerWidget extends StatefulWidget {
  final String imageSrc;
  final VoidCallback brightnessCallback;

  FlowerWidget({Key key, this.imageSrc,
this.brightnessCallback})
```

```
    : super(key: key) {
    debugPrint("FlowerWidget - constructor - " +
hashCode.toString());
  }

  @override
  _FlowerWidgetState createState() {
    debugPrint("FlowerWidget - createState - " +
hashCode.toString());
    return _FlowerWidgetState();
  }
}

class _FlowerWidgetState extends State<FlowerWidget> {
  double _blur = 0;

  _FlowerWidgetState() {
    debugPrint("_FlowerWidgetState - constructor - " +
hashCode.toString());
  }

  @override
  initState() {
    debugPrint("_FlowerWidgetState - initState - " +
hashCode.toString());
  }

  /**
   * Fired when Flutter detects that the data from another
```

```
source has changed,
   * possibly affecting the UI and causing a call to ‘build’.
   * In this case it is when the Theme changes (its an InheritedWidget).
   */
  @override
  void didChangeDependencies() {
    debugPrint(
        "_FlowerWidgetState - didChangeDependencies - " + hashCode.toString());
  }

  @override
  /**
   * Fired when the widget is reconstructed as its widget data has changed,
   * In this case it is when a new FlowerWidget is created with a different
   * imageSrc.
   */
  void didUpdateWidget(Widget oldWidget) {
    debugPrint("_FlowerWidgetState - didUpdateWidget - " + hashCode.toString());

    // The flower image has changed, so reset the blur.
    _blur = 0;
  }

  void _blurMore() {
```

```
    setState(() {
      _blur += 5.0;
    });
  }

  @override
  Widget build(BuildContext context) {
    debugPrint("_FlowerWidgetState - build - " +
hashCode.toString());
    return Scaffold(
      appBar: AppBar(title: Text("Flower"), actions: [
        new IconButton(
            icon: new Icon(Icons.refresh),
            onPressed: () {
              widget.brightnessCallback();
            })
      ]),
      body: new Container(
        decoration: new BoxDecoration(
            // dependency on inherited widget - start
            color: Theme.of(context).backgroundColor,
            // dependency on inherited widget - end
            image: new DecorationImage(
                // dependency on data from widget - start
                image: NetworkImage(widget.imageSrc),
                // dependency on data from widget - end
                fit: BoxFit.cover)),
```

```
        child: new BackdropFilter(
          // dependency on state data - start
          filter: new ImageFilter.blur(sigmaX: _blur, sigmaY:
_blur),
          // dependency on state data - end
          child: new Container(
            decoration: new BoxDecoration(color:
Colors.white.withOpacity(0.0)),
          ),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _blurMore,
        tooltip: 'Blur More',
        child: Icon(Icons.add),
      ),
    );
  }
}
```

18. Basic Material Widgets

Introduction

We are going to spend the next few chapters going over Flutter widgets and examples of their use. Reading the example source code may be difficult at this stage because we have not covered all of the techniques used in the examples, for example State Management. However, if you keep going it will all make sense eventually.

The purpose of this chapter is to introduce some of the more commonly used Flutter Widgets along with some example code that uses them.

Text

The Text widget displays a string of text with single style. Multiple line texts are allowed.
To style the entire text in one way, specify a ‘style’ property in the constructor of the Text Widget.

Notes

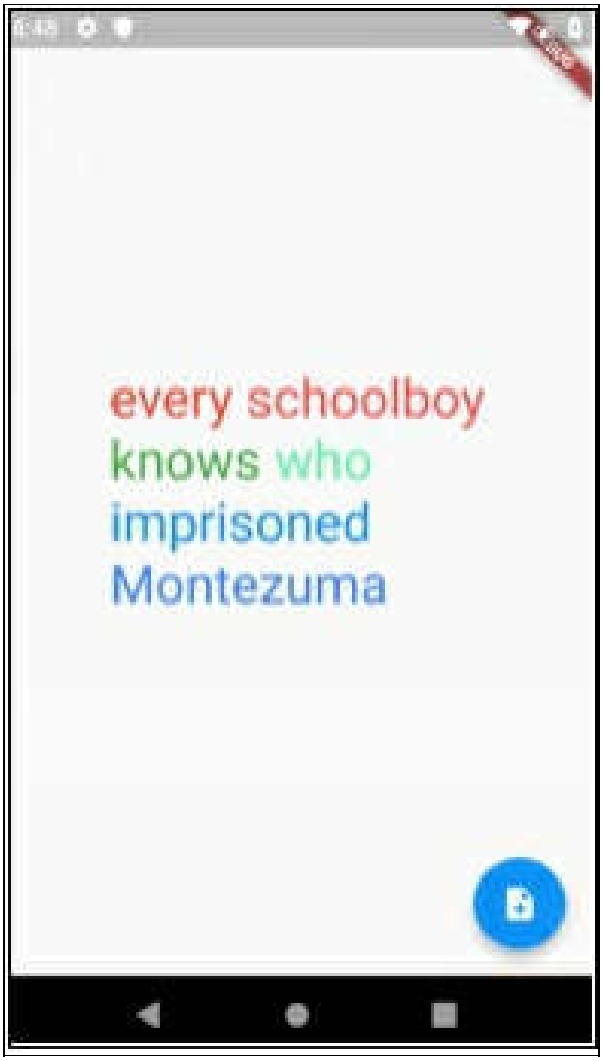
- To style sections of the text, use child TextSpans (see example below).
- The Text widget only takes up as much space

as it needs.

- If you add a Text widget, align it to centered then view it, it may not show the text as centered because the Text widget hasn’t taken up all the available horizontal space.
  - To fix this, wrap the text in a Container Widget with the width set to double.infinity.

Example – ‘text’

Every time you hit the ‘+’ a new word comes out in a different color.



Source Code

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
```

```
    return new MaterialApp(
      title: 'Flutter Styled Text Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new MyHomePage(),
    );
  }
}

class TextBlock {
  final Color _color;
  final String _text;

  TextBlock(this._color, this._text);

  String get text => _text;

  Color get color => _color;
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  _MyHomePageState createState() => new
_MyHomePageState();
}
```

```
class _MyHomePageState extends State<MyHomePage> {
  int _index = 0;
  final List<TextBlock> textBlocks = [
    TextBlock(Colors.red, 'every'),
    TextBlock(Colors.redAccent, ' schoolboy'),
    TextBlock(Colors.green, '\nknows'),
    TextBlock(Colors.greenAccent, ' who'),
    TextBlock(Colors.blue, '\nimprisoned'),
    TextBlock(Colors.blueAccent, '\nMontezuma')
  ];

  void _incrementCounter() {
    setState(() {
      if (_index < textBlocks.length) {
        _index++;
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    final List<TextSpan> textSpans = List<TextSpan>();
    for (var i = 0; i < _index; i++) {
      TextBlock textBlock = textBlocks[i];
      textSpans.add(TextSpan(
          text: textBlock.text,
          style: TextStyle(color: textBlock.color, fontSize:
```

```
32.0)));
    }
    return new Scaffold(
      body: new Center(
        child: new Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[Text.rich(TextSpan(children:
textSpans))],
        ),
      ),
      floatingActionButton: new FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: new Icon(Icons.note_add),
      ), // This trailing comma makes auto-formatting nicer for
build methods.
    );
  }
}
```

Image

Introduction

This is a widget used to show an image. When displaying an image, you specify the image source in the constructor:

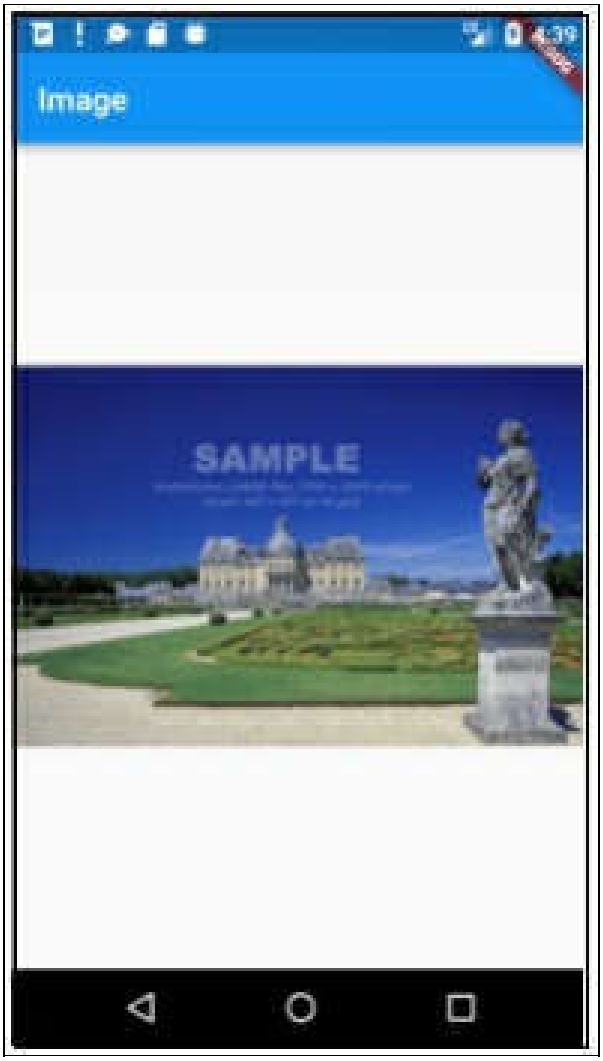
- image provider
- asset
- network
- file
- memory

The downside of the Image widget is the lack of placeholder (for example ‘loading…’ text). It shows nothing then shows the image. This doesn’t really cut it, so you need to use the FadeInImage to wrap this Widget.

The Flutter Image Widget has a fit property will enables developers to determine how the image graphics are fitted into the available area. This fit property can really change how the image is presented! See the BoxFit class documentation here: https://docs.flutter.io/flutter/painting/BoxFit-class.html

Exercise – ‘loading_image’

Load a large into an app. Display an image placeholder while it loads.



Step 1 – Create Default Flutter App

Follow the instructions in Generate Your First App
Leave project open.

Step 2 – Get Loading Image

- Download: https://digitalsynopsis.com/wp-content/uploads/2016/06/loading-animations-preloader-gifs-ui-ux-effects-10.gif

- Create new folder ‘assets’ in your project.
- Rename image file to ‘loading.gif’.
- Copy image file into ‘assets’ folder in your project.

Step 3 – Include the Loading Image in Your Project as an Asset

When you add a reference to an asset (an image or something similar) in the pubspec file, Flutter ensures that it is bundled into the app when it builds. So no round trip will be required to load the image.

- Edit the pubspec.yaml file and change the lines below from:

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

- to:

```
  assets:
  - assets/loading.gif
```

Step 4 – Replace Application Code

Replace contents of file ‘main.dart’ in folder ‘lib’ with

the following:

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

void main() => runApp(new LoadingImageApp());

class LoadingImageApp extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
  return new MaterialApp(
   title: 'Image',
   theme: new ThemeData(
    primarySwatch: Colors.blue,
   ),
   home: new HomeWidget(),
  );
 }
}

class HomeWidget extends StatelessWidget {
 HomeWidget({Key key}) : super(key: key);

  @override
 Widget build(BuildContext context) {
  return new Scaffold(
     appBar: new AppBar(
      title: new Text("Image"),
     ),
     body: new Center(
       child: FadeInImage.assetNetwork(
      placeholder: 'assets/loading.gif',
      image:
        'http://archivision.com/educational/samples/files/1A2-
F-P-I-2-C1_L.jpg',
     )));
 }
}
```

Step 5 – Open Emulator & Run

Follow the instructions in Open Android Emulator & Run Your First App
When you run this example, you see a loading icon (which very quickly goes away) then a computer.

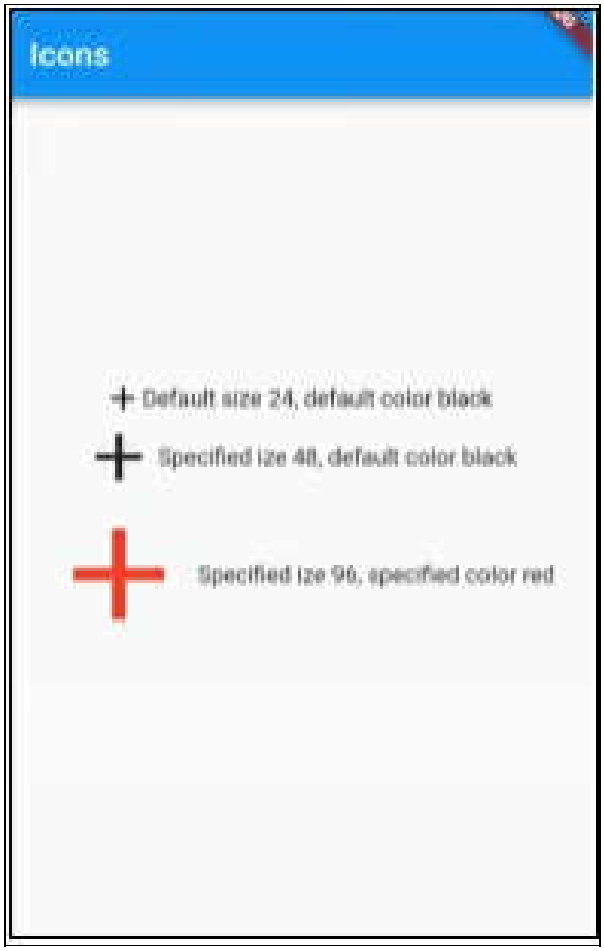
Icon

Introduction

The icon widget allows you to quickly build icon widgets using a pre-built list of material icons, available in the Icons class. You can specify the icon size and color.

Example - ‘icon’

This app simply displays 3 icons with different sizes and colors.



Source Code

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

void main() => runApp(new IconApp());

class IconApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
```

```
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    Row row1 = Row(
        mainAxisAlignment: MainAxisAlignment.center, //
center horizontally
        children: <Widget>[
          const Icon(Icons.add),
          const Text("Default size 24, default color black")
        ]);
    Row row2 =
        Row(mainAxisAlignment: MainAxisAlignment.center, //
center horizontally
            children: <Widget>[
          const Icon(Icons.add, size: 48.0),
          const Text("Specified ize 48, default color black")
        ]);
    Row row3 =
        Row(mainAxisAlignment: MainAxisAlignment.center, //
center horizontally
```

```
          children: <Widget>[
        const Icon(Icons.add, size: 96.0, color: Colors.red),
        const Text("Specified size 96, specified color red")
      ]);
    return new Scaffold(
      appBar: new AppBar(title: const Text("Icons")),
      body: new Column(
        mainAxisAlignment: MainAxisAlignment.center, //
center vertically
        children: <Widget>[row1, row2, row3]));
  }
}
```

Further Reading

You can use tools available on the internet to build your own icon library, with constants available (similar to the Icons constants). Here is a link to the article: https://steemit.com/utopian-io/@psyanite/how-to-use-custom-icons-in-flutter

Buttons

Introduction

Flutter offers a bunch of different button widgets:

- FlatButton - material

    - Useful for buttons that don’t need a border, for example those that are already in a toolbar or menu (something that provides a ui context).
    - Flashes background when clicked on.
- RaisedButton - Material
    - Useful if you want a button made more visible in a ‘sea of content’.
    - Flashes shadow when clicked on.
- IconButton - material
    - Flashes background circle when clicked on.
- OutlineButton - material
    - A bordered button whose elevation increases and whose background becomes opaque when the button is pressed.
    - Flashes background and border when clicked on.
- DropdownButton - material
    - Used for selecting from a list of items
    - Shows menu when clicked on.
    - You can supply existing value as constructor argument.
- BackButton
    - An IconButton setup for use as a back button.

    - Flashes background circle when clicked on.
- CloseButton
    - An IconButton setup for use as a close button to close modals (or any other closeable content).
    - Flashes background circle when clicked on.
- FloatingActionButton - material
    - A button that hovers in a layer above content.
    - Advisable that you only ever use one at a time.
    - You can change background and foreground colors.
    - You can use the ‘extended’ named constructor to make a larger, wider Floating Action Button.

```
FloatingActionButton.extended(
  onPressed: () {},
  icon: Icon(Icons.save),
  label: Text("Save"),
)
```

    - Flashes when clicked on.

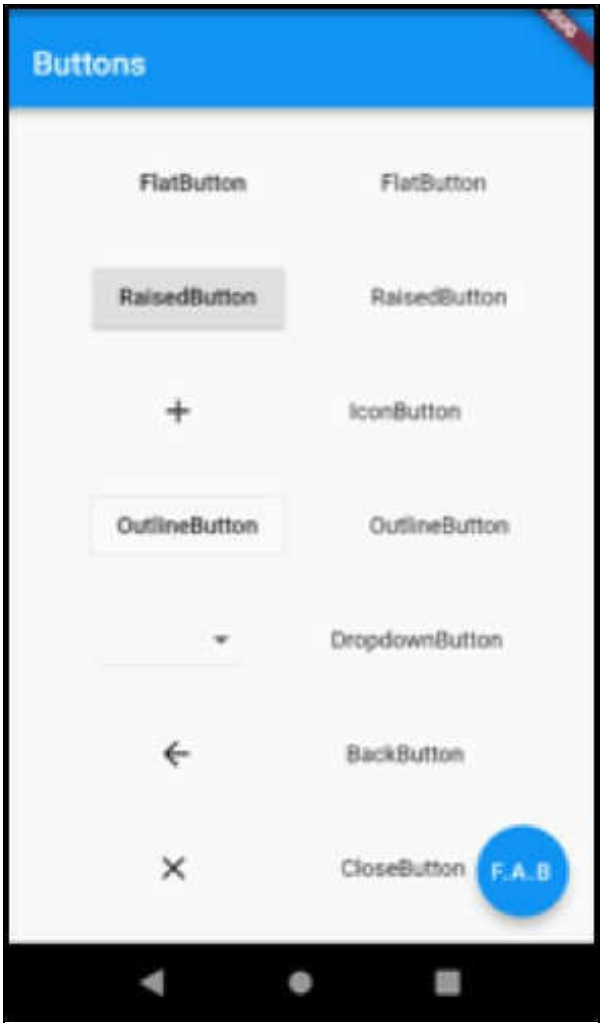
Enabling

You can enable or disable buttons using the ‘onPressed’ constructor argument.
Setting it to null disables the button, otherwise it is enabled.
The code below uses a ternary operator for this.

```
OutlineButton(
onPressed: _enabled ? _onPressed : null,
  child: const Text('Register'),
)
```

Example – ‘buttons’

This app displays different types of buttons so you can see what they look like.



Source Code

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

void main() => runApp(new ButtonApp());

class ButtonApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
```

```
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const HomeWidget(),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    Row flatButtonRow = Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          FlatButton(
              onPressed: () => debugPrint('FlatButton pressed'),
              child: Text('FlatButton')),
          const Text("FlatButton")
        ]);
    Row raisedButtonRow = Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          RaisedButton(
              onPressed: () => debugPrint('RaisedButton pressed'),
```

```
            child: Text('RaisedButton')),
          const Text("RaisedButton")
        ]);
    Row iconButtonRow = Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          IconButton(
              icon: Icon(Icons.add),
              onPressed: () => debugPrint('IconButton pressed')),
          const Text("IconButton")
        ]);
    Row outlineButtonRow = Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          OutlineButton(
              onPressed: () => debugPrint('OutlineButton
pressed'),
              child: Text("OutlineButton")),
          const Text("OutlineButton")
        ]);
    Row dropdownButtonRow = Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          new DropdownButton<String>(
              items: <String>['Mens', 'Womans'].map((String
value) {
                return new DropdownMenuItem<String>(
```

```
                value: value,
                child: Text(value),
              );
            }).toList(),
            onChanged: (value) => debugPrint('Changed: ${value}')),
        const Text("DropdownButton")
      ]);

  Row backButtonRow = Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: <Widget>[BackButton(), const Text("BackButton")]);

  Row closeButtonRow = Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: <Widget>[CloseButton(), const Text("CloseButton")]);

  return new Scaffold(
    appBar: new AppBar(
      title: const Text("Buttons"),
    ),
    body: new Center(
      child: new Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          flatButtonRow,
```

```
            raisedButtonRow,
            iconButtonRow,
            outlineButtonRow,
            dropdownButtonRow,
            backButtonRow,
            closeButtonRow,
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
          onPressed: () => debugPrint('FloatingActionButton
pressed'),
          child: const Text("F.A.B")),
    );
  }
}
```

19. Multi-Child Layout Widgets

Introduction

Layout Widgets are non-visible Widgets that are used that affect the positioning and presentation of multiple child widgets.

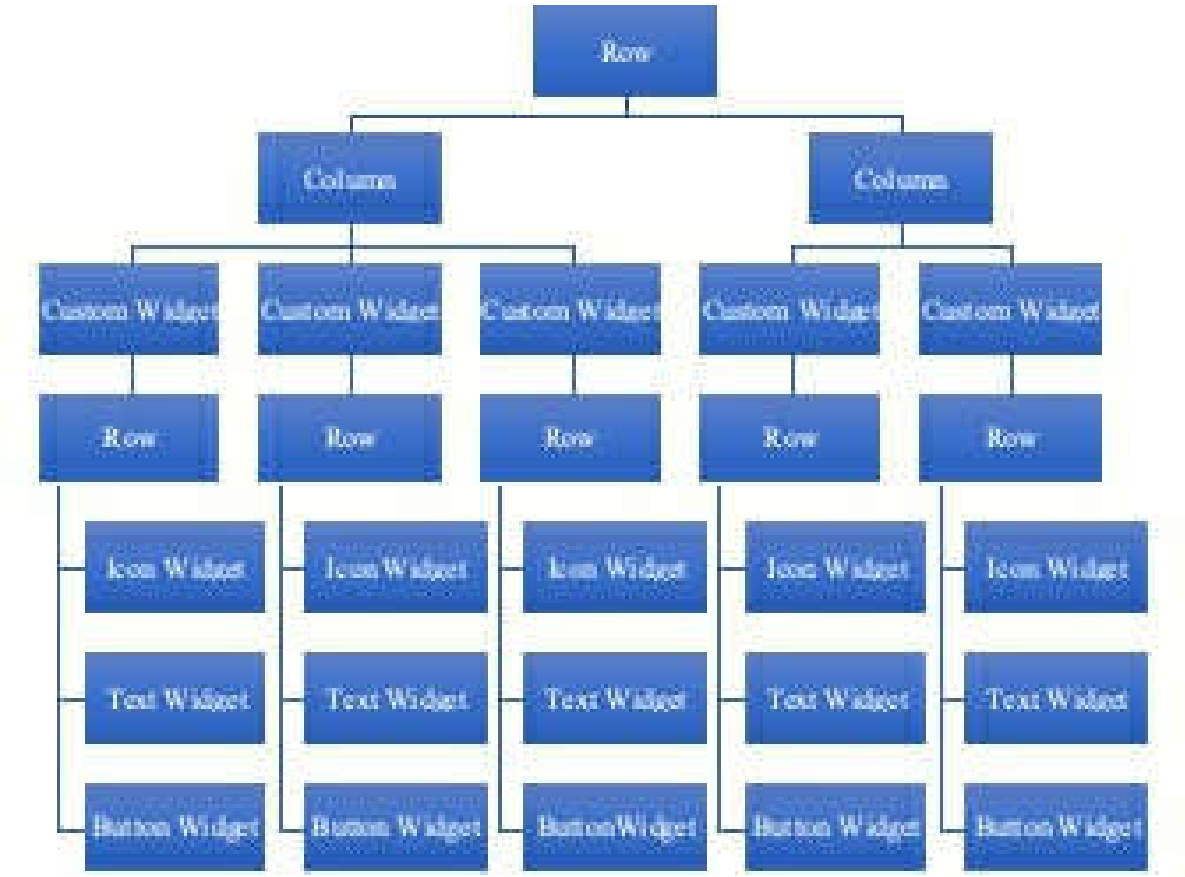
There are two main kinds of Layout Widgets: Single-Child Layout Widgets and Multi-Child Layout Widgets.

The purpose of this chapter is to cover Multi-Child Layout Widgets.

Multi-Child Layout Widgets

Multi-Child Layout Widgets and they are used to determine what UI elements go where - where the elements of the user interface are going to be presented. They are very important as you can break almost 90% of the layout designs into Rows and Columns.

Obviously, you can combine/nest these Widgets. You could have a Row that contains 2 Columns that contains 3 Custom Widgets. Then each Custom Widget could contain a Row of an Icon Widget, a Text Widget then a Button.



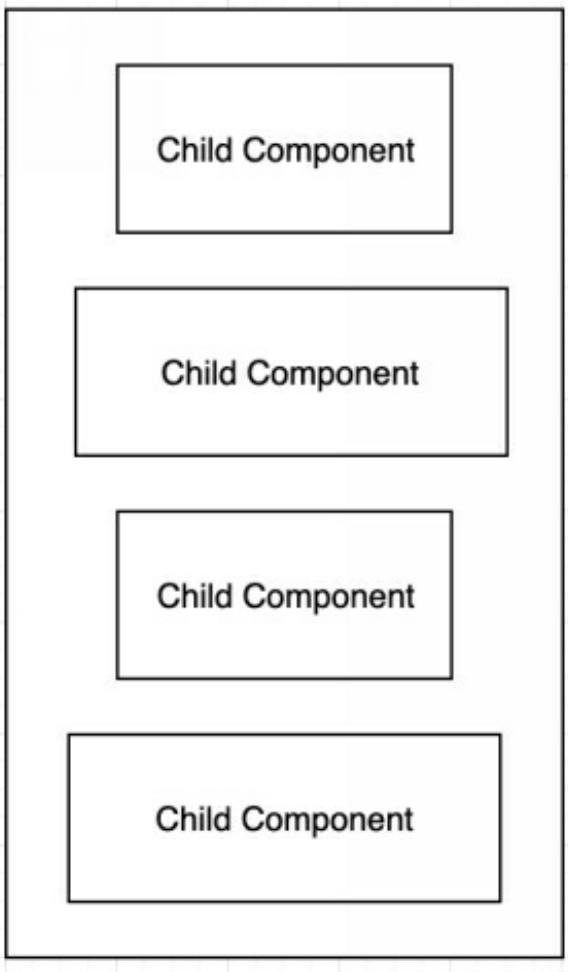
These layouts work really well when they are used to layout components to which you already know the size, such as buttons, textboxes etc. They also work when you have widgets that don't overflow the screen space available and you expand them to use all the space available up. When you need to use up extra screen space you can use the MainAxisAlignment property to space child Widgets out or use Expanded Widgets to expand those child Widgets.

These layouts don't work well when they are used to layout components with very dynamic sizing

requirements, for example Text widgets that are generated from user data, with some wide texts, some narrow texts. In this case, you are probably better off using the Table. It can handle the text overflows without any additional complications.

When using these Widgets, you may sometimes encounter the times when the child Widgets don’t fit in the screen space. This often results in visible chevrons (the yellow and black stripes) such as you see below, along with a console error:

Column



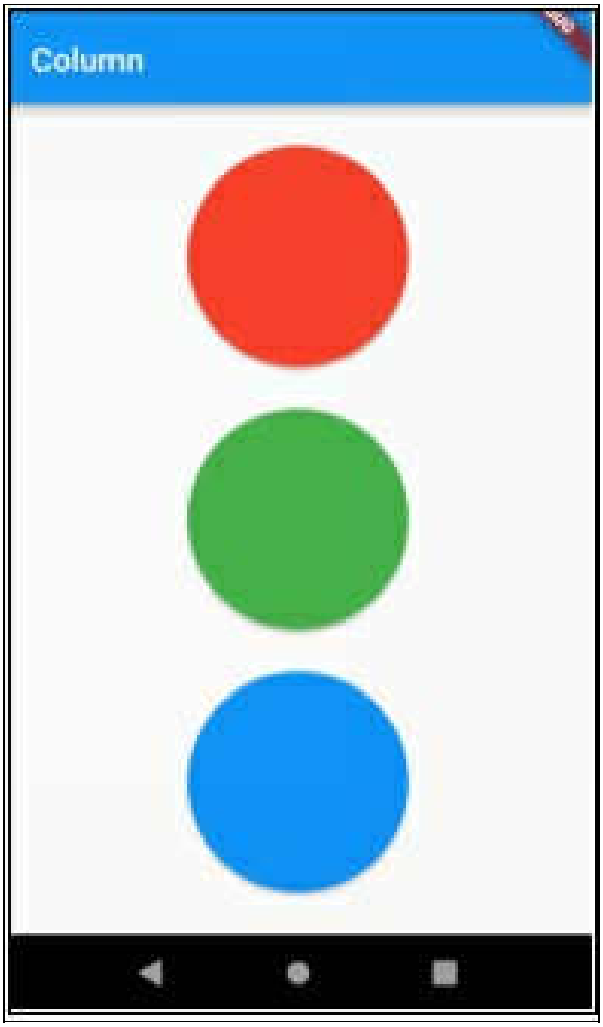
- Enables you to lay out Widgets Vertically.
- Use the MainAxisAlignment to specify vertical layout.
- Use the CrossAxisAlignment to specify horizontal alignment.
- Will try to take up as much space as it needs for children but no more.
    - To use all available space, wrap in Expanded widget.

- Does not provide scrolling.
    - If you run out of vertical space, you may get an error.
    - If you need to include scrolling, use a ListView instead.

Spacing Out Children Using MainAxisAlignment

The MainAxisAlignment widget allows you to determine how the Widgets are laid out vertically. Take a look at the example below to see how this affects the horizontal layouts.

Example – ‘column_spaced_evenly’



Source Code

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

void main() => runApp(new ColumnSpacedEvenly());

class ColumnSpacedEvenly extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
```

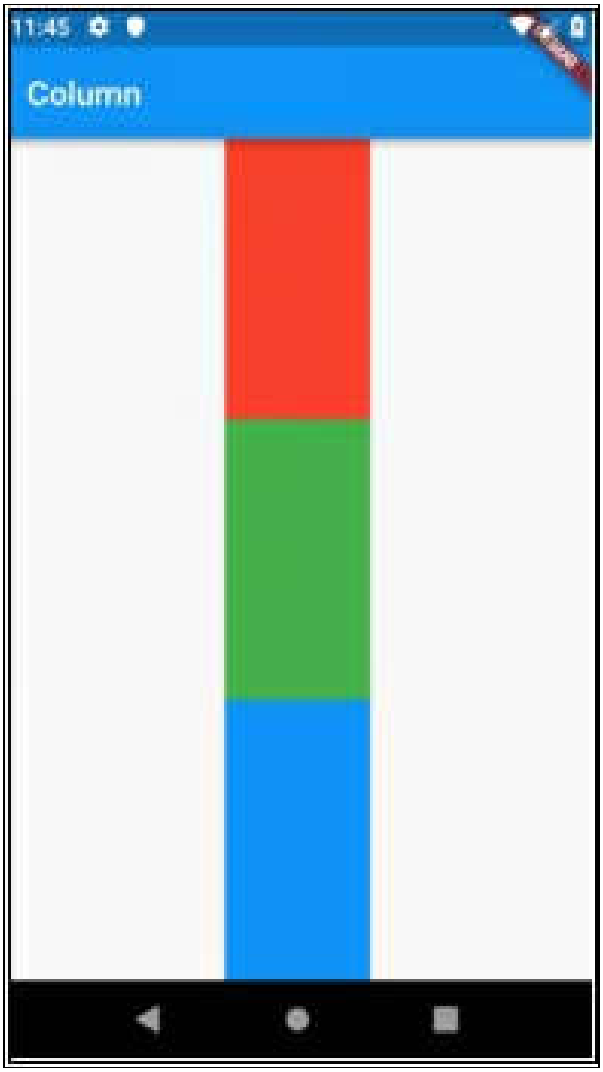
```
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatelessWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    RawMaterialButton redButton = RawMaterialButton(
      constraints: const BoxConstraints(minWidth: 188.0,
minHeight: 136.0),
      onPressed: () {},
      shape: new CircleBorder(),
      elevation: 2.0,
      fillColor: Colors.red,
      padding: const EdgeInsets.all(15.0),
    );
    RawMaterialButton greenButton = new RawMaterialButton(
      constraints: const BoxConstraints(minWidth: 188.0,
minHeight: 136.0),
      onPressed: () {},
      shape: new CircleBorder(),
```

```
      elevation: 2.0,
      fillColor: Colors.green,
      padding: const EdgeInsets.all(15.0),
    );
    RawMaterialButton blueButton = new RawMaterialButton(
      constraints: const BoxConstraints(minWidth: 188.0,
minHeight: 136.0),
      onPressed: () {},
      shape: new CircleBorder(),
      elevation: 2.0,
      fillColor: Colors.blue,
      padding: const EdgeInsets.all(15.0),
    );
    return new Scaffold(
        appBar: new AppBar(
          title: new Text("Column"),
        ),
        body: new Center(
          child: new Column(
            mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
            children: <Widget>[redButton, greenButton,
blueButton],
          ),
        ));
  }
}
```

Expanding Children Using Expanded Widget

If you use an Expanded Widget (Single-Child Layout Widget) around each of your child Widgets, this allows them to expand to fit the available space.

Example – ‘column_expanded’

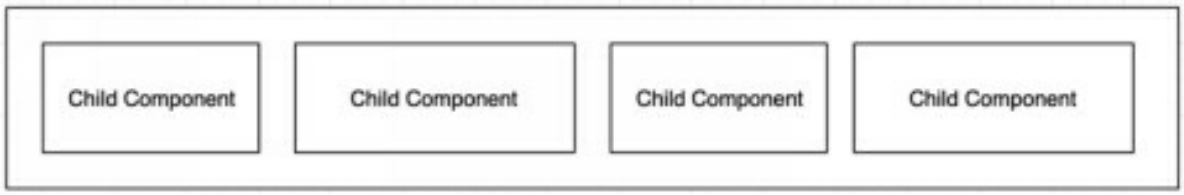


Source Code

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

void main() => runApp(new ColumnSpacedEvenly());

class ColumnSpacedEvenly extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatelessWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    RawMaterialButton redButton = RawMaterialButton(
        onPressed: () {}, elevation: 2.0, fillColor: Colors.red);
    RawMaterialButton greenButton = new RawMaterialButton(
      onPressed: () {},
```

```
    elevation: 2.0,
    fillColor: Colors.green,
  );
  RawMaterialButton blueButton = new RawMaterialButton(
    onPressed: () {},
    elevation: 2.0,
    fillColor: Colors.blue,
  );
  return new Scaffold(
      appBar: new AppBar(
        title: new Text("Column"),
      ),
      body: new Center(
        child: new Column(
          mainAxisAlignment: MainAxisAlignment.start,
          children: <Widget>[
            Expanded(child: redButton),
            Expanded(child: greenButton),
            Expanded(child: blueButton)
          ],
        ),
      ));
 }
}
```

Row



- Enables you to lay out Widgets Horizontally.
- Use the MainAxisAlignment to specify layout.

- If you run out of horizontal space, you may get an error and chevrons may appear.

Spacing Out Children Using MainAxisAlignment

The MainAxisAlignment widget allows you to determine how the Widgets are laid out horizontally. Take a look at the example below to see how this affects the horizontal layouts.

Example – ‘row_main_axis_alignment’


Rows
MainAxisAlignment.center
MainAxisAlignment.end
MainAxisAlignment is spaceAround
MainAxisAlignment is spaceEvenly
MainAxisAlignment is spaceBetween
MainAxisAlignment.start

Source Code

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

void main() => runApp(new RowMainAxisAlignmentApp());

class RowMainAxisAlignmentApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatelessWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(title: new Text("Rows")),
        body: new Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            Row(
```

```
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text("MainAxisAlignment"),
            const Text("is"),
            const Text("center")
          ],
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.end,
          children: <Widget>[
            const Text("MainAxisAlignment"),
            const Text("is"),
            const Text("end")
          ],
        ),
        Row(
          mainAxisAlignment:
MainAxisAlignment.spaceAround,
          children: <Widget>[
            const Text("MainAxisAlignment"),
            const Text("is"),
            const Text("spaceAround")
          ],
        ),
        Row(
          mainAxisAlignment:
```

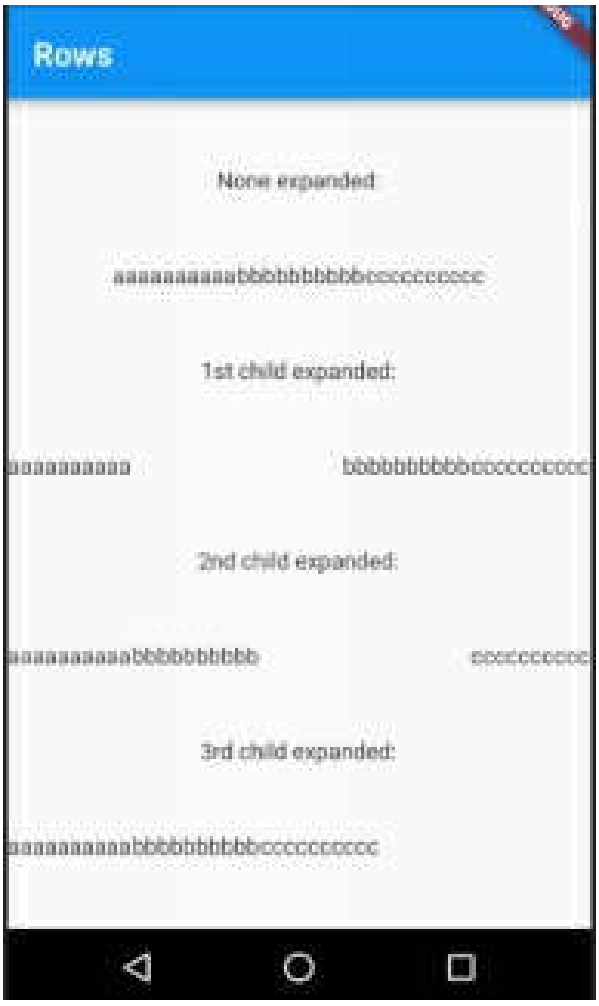
```
MainAxisAlignment.spaceEvenly,
            children: <Widget>[
              const Text("MainAxisAlignment"),
              const Text("is"),
              const Text("spaceEvenly")
            ],
          ),
          Row(
            mainAxisAlignment:
MainAxisAlignment.spaceBetween,
            children: <Widget>[
              const Text("MainAxisAlignment"),
              const Text("is"),
              const Text("spaceBetween")
            ],
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.start,
            children: <Widget>[
              const Text("MainAxisAlignment"),
              const Text("is"),
              const Text("start")
            ],
          ),
        ],
      ));
```

```
  }
}
```

Expanding Children Using Expanded Widget

If you use an Expanded Widget (Single-Child Layout Widget) around some of your child Widgets, that allows them to expand to fit the available space.

Example – ‘row_with_expanded’



Source Code

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

void main() => runApp(new RowWithExpandedApp());

class RowWithExpandedApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatelessWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(title: new Text("Rows")),
        body: new Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
```

```
children: <Widget>[
  const Text("None expanded:"),
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      const Text("aaaaaaaaaa"),
      const Text("bbbbbbbbbb"),
      const Text("cccccccccc")
    ],
  ),
  const Text("1st child expanded:"),
  Row(
    children: <Widget>[
      const Expanded(child: const Text("aaaaaaaaaa")),
      const Text("bbbbbbbbbb"),
      const Text("cccccccccc")
    ],
  ),
  const Text("2nd child expanded:"),
  Row(
    children: <Widget>[
      const Text("aaaaaaaaaa"),
      const Expanded(child: const Text("bbbbbbbbbb")),
      const Text("cccccccccc")
    ],
  ),
```

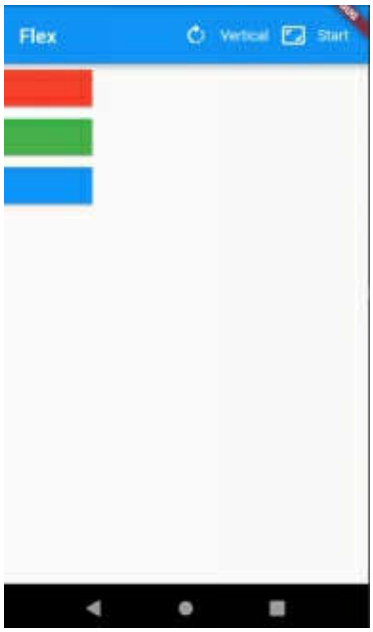
```
          const Text("3rd child expanded:"),
          Row(
            children: <Widget>[
              const Text("aaaaaaaaaa"),
              const Text("bbbbbbbbbb"),
              const Expanded(child: const Text("ccccccccccc"))
            ],
          ),
        ],
      ));
  }
}
```

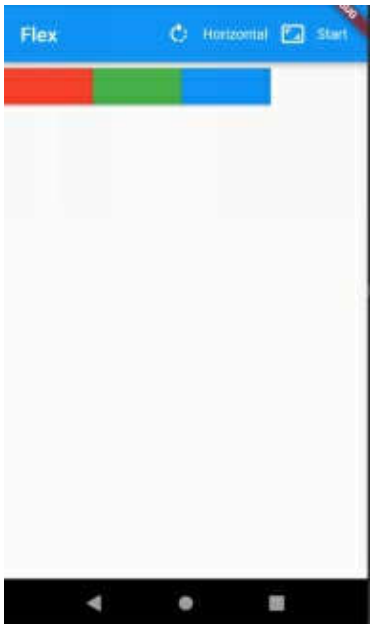
Flex

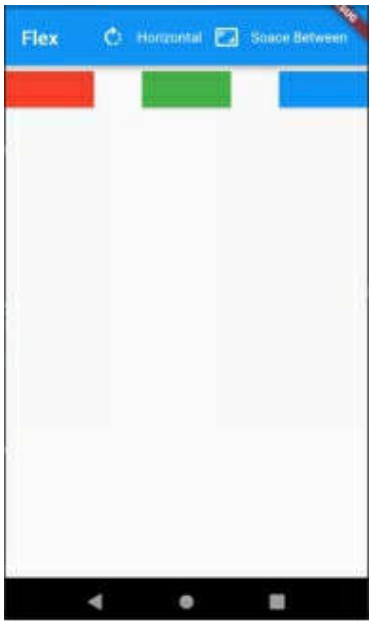
The Flex Widget is similar to Row and Column widget, except that it can act as both when you specify the mainAxis.

Example – ‘flex’

This app uses the Flex layout for the main content – three rectangles. It has a toolbar with two buttons. The first button allows the user to toggle the Flex axis between vertical and horizontal. The second button allows the user to change the value of the main axis alignment.







Source Code

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
        title: 'Flutter Demo',
        theme: new ThemeData(
          primarySwatch: Colors.blue,
        ),
        home: new HomeWidget(title: 'Flex'));
  }
}
```

```
class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => new
_MyHomePageState();
}

class _MyHomePageState extends State<HomeWidget> {
  List<MainAxisAlignment> _alignments = [
    MainAxisAlignment.start,
    MainAxisAlignment.end,
    MainAxisAlignment.center,
    MainAxisAlignment.spaceBetween,
    MainAxisAlignment.spaceEvenly,
    MainAxisAlignment.spaceAround
  ];
  List<String> _alignmentsText = [
    "Start",
    "End",
    "Center",
    "Soace Between",
    "Space Evenly",
    "Space Around"
  ];
```

```
bool _vertical = true;
int _alignmentIndex = 0;

RawMaterialButton redButton = RawMaterialButton(
    onPressed: () {}, elevation: 2.0, fillColor: Colors.red);
RawMaterialButton greenButton = new RawMaterialButton(
  onPressed: () {},
  elevation: 2.0,
  fillColor: Colors.green,
);
RawMaterialButton blueButton = new RawMaterialButton(
  onPressed: () {},
  elevation: 2.0,
  fillColor: Colors.blue,
);

@override
Widget build(BuildContext context) {
  return new Scaffold(
      appBar: new AppBar(
        title: new Text(widget.title),
        actions: <Widget>[
          IconButton(
            icon: const Icon(Icons.rotate_right),
            tooltip: 'Direction',
            onPressed: () {
              setState(() {
```

```
          _vertical = !_vertical;
        });
      },
    ),
    Padding(
        padding: EdgeInsets.only(top: 20.0),
        child: Text(_vertical ? "Vertical" : "Horizontal")),
    IconButton(
      icon: const Icon(Icons.aspect_ratio),
      tooltip: 'Main axis',
      onPressed: () {
        setState(() {
          _alignmentIndex++;
          if (_alignmentIndex >= _alignments.length) {
            _alignmentIndex = 0;
          }
        });
      },
    ),
    Padding(
        padding: EdgeInsets.only(top: 20.0),
        child: Text(_alignmentsText[_alignmentIndex])),
    Padding(
      padding: EdgeInsets.all(10.0),
    )
  ],
```

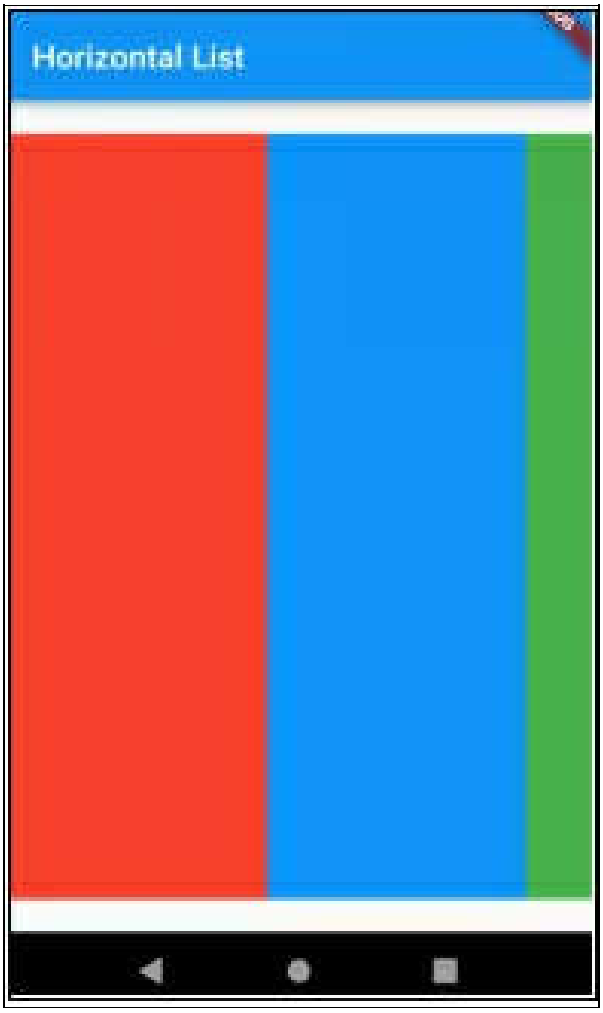
```
        ),
        body: new Flex(
          direction: _vertical ? Axis.vertical : Axis.horizontal,
          mainAxisAlignment: _alignments[_alignmentIndex],
          children: <Widget>[redButton, greenButton,
blueButton],
        ));
  }
}
```

ListView & ListTiles

The ListView Widget is similar to the Flex widget in that it can act as both a horizontal list and a vertical list. The difference is that it provides scrolling out of the box.

Example - ‘horizontal_list’

This app displays a list of Widgets horizontally rather than vertically. You can scroll by swiping to the left or to the right.



Source Code

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

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

class HorizontalListApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final title = 'Horizontal List';
    return MaterialApp(
```

```
title: title,
home: Scaffold(
  appBar: AppBar(
    title: Text(title),
  ),
  body: Container(
    margin: EdgeInsets.symmetric(vertical: 20.0),
    child: ListView(
      scrollDirection: Axis.horizontal,
      children: <Widget>[
        Container(
          width: 160.0,
          color: Colors.red,
        ),
        Container(
          width: 160.0,
          color: Colors.blue,
        ),
        Container(
          width: 160.0,
          color: Colors.green,
        ),
        Container(
          width: 160.0,
          color: Colors.yellow,
        ),
```

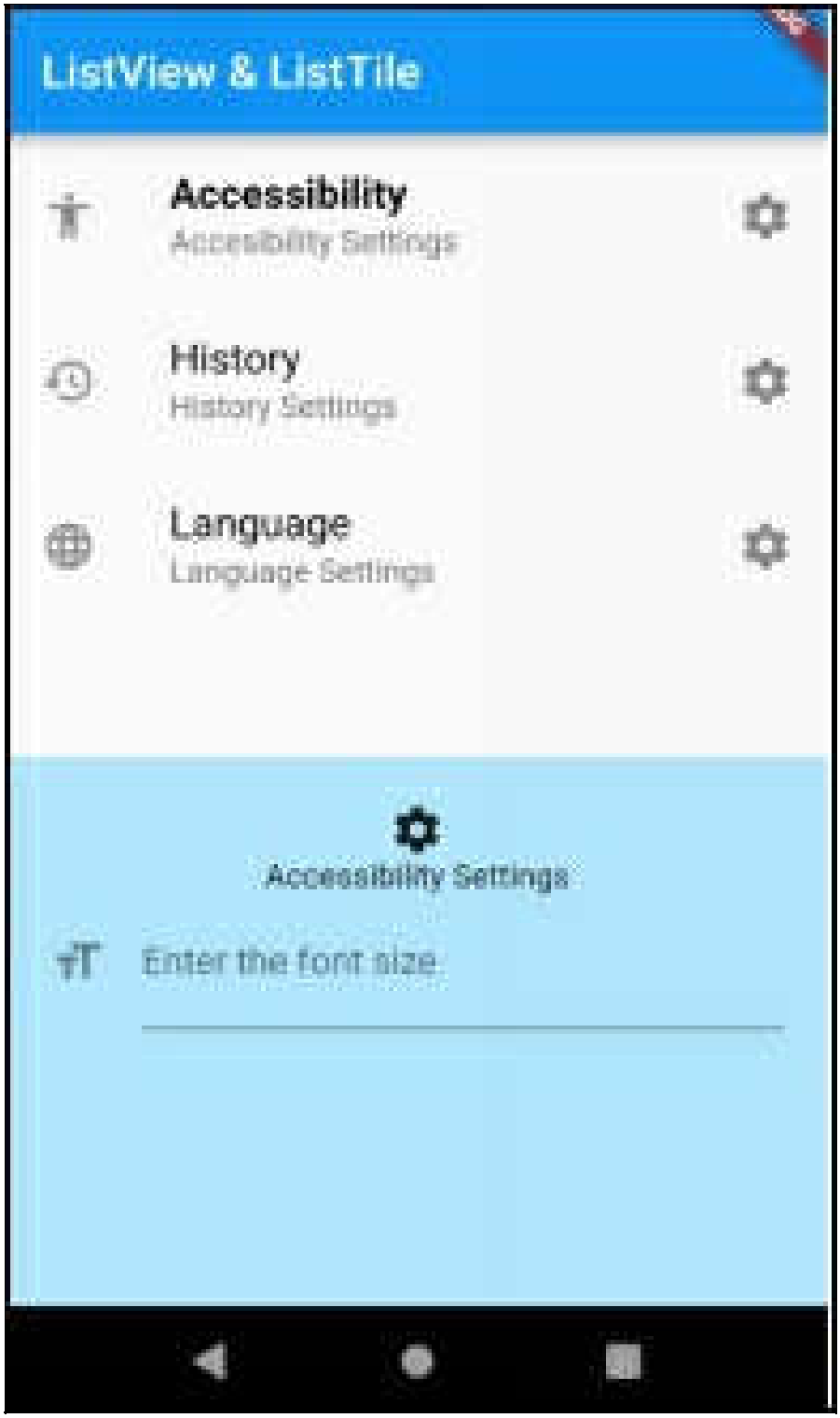
```
            Container(
              width: 160.0,
              color: Colors.orange,
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

ListTile

A list tile contains one to three lines of text optionally flanked by icons or other widgets, such as check boxes. So, you can have text in the middle and a widget on each side. Here is an example of a ListTile:

Many people combine ListViews and ListTiles together because ListTiles are great for building great-looking selection lists.

Example – ‘settings’


ListView & ListTile
Accessibility
Accessibility Settings
History
History Settings
Language
Language Settings
Accessibility Settings
Enter the font size

Source Code

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

void main() => runApp(new ListViewListTileApp());

class ListViewListTileApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'ListView & ListTile'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
```

```
  int _selectedIndex = 0;
  static const TEXT_STYLE_NORMAL = const TextStyle(
      color: Colors.black, fontSize: 18.0, fontWeight:
FontWeight.normal);
  static const TEXT_STYLE_SELECTED = const TextStyle(
      color: Colors.black, fontSize: 18.0, fontWeight:
FontWeight.bold);
  final TextFormField _fontSizeTextField = TextFormField(
      decoration: InputDecoration(
          icon: const Icon(Icons.format_size),
          hintText: 'Font Size',
          labelText: 'Enter the font size'));
  final TextFormField _historyTextFormField =
TextFormField(
      decoration: InputDecoration(
          icon: const Icon(Icons.history),
          hintText: 'Days',
          labelText: 'Enter days'));
  final TextFormField _languageTextFormField =
TextFormField(
      decoration: InputDecoration(
          icon: const Icon(Icons.language),
          hintText: 'Language',
          labelText: 'Enter your language'));

  select(index) {
    setState(() {
```

```
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    final ListTile accessibilityListTile = ListTile(
        leading: Icon(Icons.accessibility),
        title: Text("Accessibility",
            style:
                _selectedIndex == 0 ? TEXT_STYLE_SELECTED :
TEXT_STYLE_NORMAL),
        subtitle: const Text("Accesibility Settings"),
        trailing: Icon(Icons.settings),
        onTap: () => select(0));

    final ListTile historyListTile = ListTile(
        leading: Icon(Icons.history),
        title: Text("History",
            style:
                _selectedIndex == 1 ? TEXT_STYLE_SELECTED :
TEXT_STYLE_NORMAL),
        subtitle: const Text("History Settings"),
        trailing: Icon(Icons.settings),
        onTap: () => select(1));

    final ListTile languageListTile = ListTile(
        leading: Icon(Icons.language),
```

```
      title: Text("Language",
          style:
              _selectedIndex == 2 ? TEXT_STYLE_SELECTED :
TEXT_STYLE_NORMAL),
      subtitle: const Text("Language Settings"),
      trailing: Icon(Icons.settings),
      onTap: () => select(2));

   final String selectionTitle = (_selectedIndex == 0
          ? "Accessibility"
          : _selectedIndex == 1 ? "History" : "Language") +
      " Settings";

   final TextFormField selectionTextFormField =
_selectedIndex == 0
      ? _fontSizeTextField
      : _selectedIndex == 1 ? _historyTextFormField :
_languageTextFormField;

   return new Scaffold(
      appBar: new AppBar(
        title: new Text(widget.title),
      ),
      body: ListView(children: <Widget>[
        accessibilityListTile,
        historyListTile,
        languageListTile
      ]),
```

```
      bottomSheet: Container(
        color: Color(0xFFB3E5FC),
        padding: EdgeInsets.all(20.0),
        child: Container(
          constraints: BoxConstraints(maxHeight: 200.0),
          child: Column(children: <Widget>[
            Icon(Icons.settings),
            Text(selectionTitle),
            Expanded(child: selectionTextFormField)
          ])));
  }
}
```

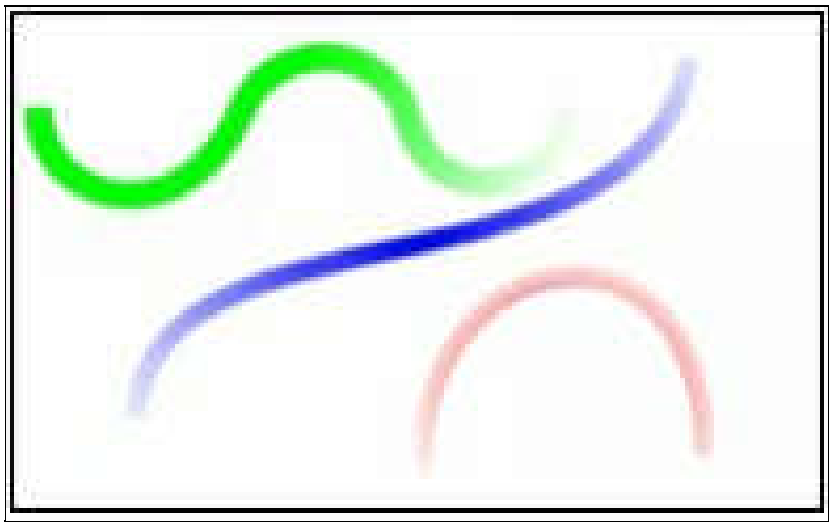
Stack

The Stack Layout Widget is useful for overlaying Widgets on top of each other. Each child of a Stack Layout Widget is either positioned or non-positioned. Positioned children are those wrapped in a Positioned widget that has at least one non-null property.

The stack paints its children in order with the first child being at the bottom. If you want to change the order in which the children paint, you can rebuild the stack with the children in the new order. In this case, ensure each child has a key to prevent it from being rebuilt every-time.

You can use a stack to overlay layers on top of other layers to create effects.

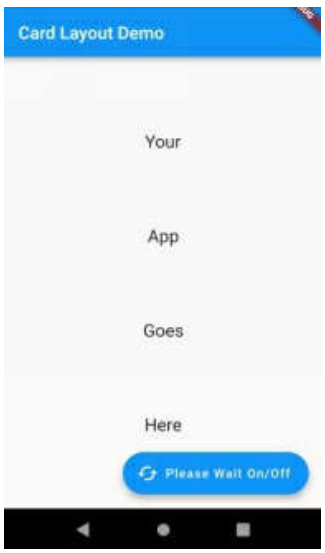
- You could have an image as a first child then add a second child that is a transparent image (ie gif or png) with a gradient from fully transparent to non-transparent. This would have the effect of ‘fading out’ the child below. Something like this:

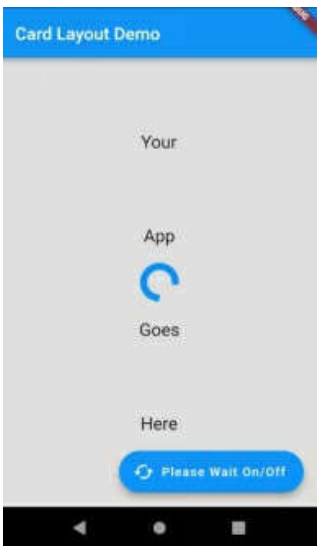


- You could use an image (or anything else) as a first child then add a second child that is a transparent image (ie gif or png) to ‘round out’ the corners, improving its appearance. Something like this:

Example – ‘stack_please_wait’

Many applications need to show a ‘please wait’ indicator which something is loading. For example, when the user logs in, the app needs to contact the server and verify your information asynchronously. This app enables the user to toggle a ‘please wait’ indicator on or off.





Source Code

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

void main() => runApp(new StackPleaseWaitAppWidget());

class StackPleaseWaitAppWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Card Layout Demo'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);
  final String title;
  final PleaseWaitWidget _pleaseWaitWidget =
      PleaseWaitWidget(key: ObjectKey("pleaseWaitWidget"));
  final AppWidget _appWidget = AppWidget(key:
ObjectKey("appWidget"));

  @override
```

```
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  bool _pleaseWait = false;

  void _togglePleaseWait() {
    setState(() {
      _pleaseWait = !_pleaseWait;
    });
  }

  @override
  Widget build(BuildContext context) {
    List<Widget> childWidgets = _pleaseWait
        ? [widget._pleaseWaitWidget, widget._appWidget]
        : [widget._appWidget];
    return new Scaffold(
        appBar: new AppBar(
          title: new Text(widget.title),
        ),
        body: new Center(
            child: Stack(key: ObjectKey("stack"), children:
childWidgets)),
        floatingActionButton: new
FloatingActionButton.extended(
            onPressed: _togglePleaseWait,
```

```
            label: Text('Please Wait On/Off'),
            icon: new Icon(Icons.cached)));
  }
}

class PleaseWaitWidget extends StatelessWidget {
  PleaseWaitWidget({
    Key key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
        child: Center(
          child: CircularProgressIndicator(strokeWidth: 8.0),
        ),
        color: Colors.grey.withOpacity(0.3));
  }
}

class AppWidget extends StatelessWidget {
  AppWidget({
    Key key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Center(
```

```
      child: new Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          const Text('Your', style: TextStyle(fontSize: 20.0)),
          const Text('App', style: TextStyle(fontSize: 20.0)),
          const Text('Goes', style: TextStyle(fontSize: 20.0)),
          const Text('Here', style: TextStyle(fontSize: 20.0))
        ],
      ),
    );
  }
}
```

20. Single-Child Layout Widgets

Introduction

Earlier we mentioned that there are two main kinds of Layout Widgets: Single-Child Layout Widgets and Multi-Child Layout Widgets. We covered Multi-Child Layout Widgets in the previous chapter.

Single-Child Layout Widgets are non-visible Widgets that are used that affect the positioning and presentation of a single child widget. The purpose of this chapter is to cover these widgets.

Most Important Single Child Layout Widgets

- The Padding Widget is probably used most of all these and is used to affect the padding around its child widget.
- The Container Widget is a Single-Child Layout Widget in that it lays out Child Widgets. However, it can be visible and you can style it.

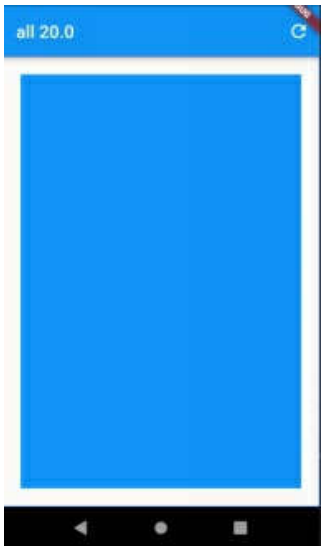
Padding

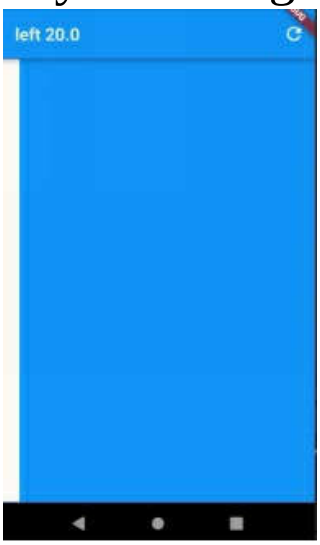
Used all the time to add padding around a child

Widget. It uses EdgeInset objects to specify the padding metrics around the child Widget.

Example – ‘padding’

This app allows the user click on an icon on the right side of the toolbar to cycle through the border insets.





Source Code

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
```

```
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  static const double TWENTY = 20.0;
  static const List<String> _titles = [
    "all 20.0",
    "left 20.0",
    "right 20.0",
    "top 20.0",
    "bottom 20.0",
    "sym horiz 20.0",
    "sym vert 20.0"
  ];
```

```
  static const List<EdgeInsets> _edgeInsets = [
    const EdgeInsets.all(TWENTY),
    const EdgeInsets.only(left: TWENTY),
    const EdgeInsets.only(right: TWENTY),
    const EdgeInsets.only(top: TWENTY),
    const EdgeInsets.only(bottom: TWENTY),
    const EdgeInsets.symmetric(horizontal: TWENTY),
    const EdgeInsets.symmetric(vertical: TWENTY)
  ];
  int _index = 0;
  final Container _childContainer = Container(color:
Colors.blue);

  void _next() {
    setState(() {
      _index++;
      if (_index >= _titles.length) {
        _index = 0;
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    Padding padding =
        Padding(padding: _edgeInsets[_index], child:
_childContainer);
```

```
    return Scaffold(
        appBar: AppBar(
          title: Text(_titles[_index]),
          actions: [
            new IconButton(
                icon: new Icon(Icons.refresh), onPressed: () =>
_next())
          ],
        ),
        body: Center(
            child: Container(
                child: padding,
                decoration: BoxDecoration(
                    border: new Border.all(color:
Colors.blueAccent)))));
  }
}
```

Container

A convenience widget that combines common painting, positioning, and sizing widgets. Often used to contain wrap child widgets and apply styling.

Example – ‘container’

This example app shows an aircraft in a container Widget with a border and a background. The user can

hit the button on the bottom right to spin the aircraft. Ignore the animation code for the moment.

Source Code

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

void main() => runApp(new ContainerApp());

class ContainerApp extends StatelessWidget {
  // This widget is the root of your application.
```

```
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Flutter Demo Home Page'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget>
    with SingleTickerProviderStateMixin {
  Animation<double> _animation;
  AnimationController _controller;

  @override
  void initState() {
```

```
    super.initState();
    _controller =
        AnimationController(duration: const Duration(seconds:
2), vsync: this);
    _animation = Tween<double>(begin: 0.0, end:
1.0).animate(_controller)
      ..addListener(() {
        setState(() {});
      });
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(
          title: new Text(widget.title),
        ),
        body: new Center(
            child: new Container(
                child: new RotationTransition(
                    turns: new
AlwaysStoppedAnimation(_animation.value),
                    child: new Icon(Icons.airplanemode_active,
size: 150.0)),
                decoration: BoxDecoration(
                    border: Border.all(width: 2.0, color:
Colors.black),
                    borderRadius:
```

```
BorderRadius.all(Radius.circular(8.0)),
                    color: Colors.redAccent))),
        floatingActionButton: new FloatingActionButton(
            onPressed: _spin,
            tooltip: 'Increment',
            child: new Icon(Icons.rotate_right)));
  }

  void _spin() {
    _controller.forward(from: 0.0);
  }
}
```

Further Reading

https://medium.com/flutter-community/flutters-container-this-ain-t-your-daddy-s-div-100817339610

Card

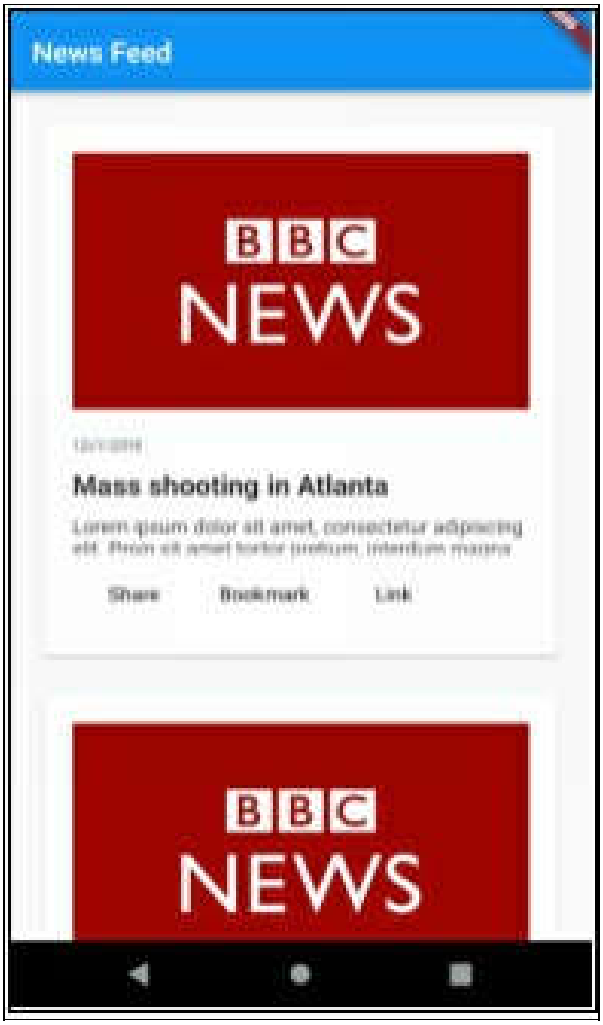
Material UI uses cards. They are used contain content and actions about a single subject.
According to the Google Documentation:

- A card is identifiable as a single, contained unit.
- A card can stand alone, without relying on surrounding elements for context.

- A card cannot merge with another card, or divide into multiple cards.

Example – ‘cards’

This app displays a news feed using Cards.
Note that this example uses a stream ‘map’ function to convert News data objects into Widgets.



Source Code

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new NewsfeedWidget(title: 'News Feed'),
    );
  }
}

class News {
  DateTime _dt;
  String _title;
  String _text;

  News(this._dt, this._title, this._text);
}

class NewsCard extends StatelessWidget {
  News _news;
```

```
  NewsCard(this._news);

  @override
  Widget build(BuildContext context) {
    return Padding(
        padding: EdgeInsets.only(bottom: 20.0),
        child: Card(
            child: Padding(
                padding: EdgeInsets.all(20.0),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: <Widget>[
                    Image.network("https://www.bbc.co"
                        ".uk/news/special/2015/newsspec_10857/bbc_n
cb=1"),
                    Padding(
                        padding: EdgeInsets.only(top: 20.0, bottom:
10.0),
                        child: Text(
                          "${_news._dt.month}//${_news._dt.day}/${_
                          style: TextStyle(
                              fontSize: 10.0, fontStyle:
FontStyle.italic),
                        )),
                    Padding(
                        padding: EdgeInsets.only(bottom: 10.0),
                        child: Text("${_news._title}",
```

```
                    style: TextStyle(
                        fontSize: 20.0, fontWeight:
FontWeight.bold))),
                Text(
                  "${_news._text}",
                  maxLines: 2,
                  style: TextStyle(fontSize: 14.0),
                  overflow: TextOverflow.fade,
                ),
                Row(children: [
                  FlatButton(child: Text("Share"), onPressed: ()
=> {}),
                  FlatButton(child: Text("Bookmark"),
onPressed: () => {}),
                  FlatButton(child: Text("Link"), onPressed: ()
=> {})
                ])
              ],
            ))));
  }
}

class NewsfeedWidget extends StatelessWidget {
  NewsfeedWidget({Key key, this.title}) : super(key: key);

  final String title;
  List<News> _newsList = [
    News(
```

```
      DateTime(2018, 12, 1),
      "Mass shooting in Atlanta",
      "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin sit amet " +
          "tortor pretium, interdum magna sed, pulvinar ligula."),
    News(
      DateTime(2019, 1, 12),
      "Carnival clown found drunk in Misisippi",
      "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin sit amet " +
          "tortor pretium, interdum magna sed, pulvinar ligula."),
    News(
      DateTime(2019, 2, 12),
      "Walrus found in family pool in Florida",
      "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin sit amet " +
          "tortor pretium, interdum magna sed, pulvinar ligula."),
  ];

  @override
  Widget build(BuildContext context) {
    List<Widget> newsCards = _newsList.map((news) => NewsCard(news)).toList();
    return new Scaffold(
      appBar: new AppBar(
```

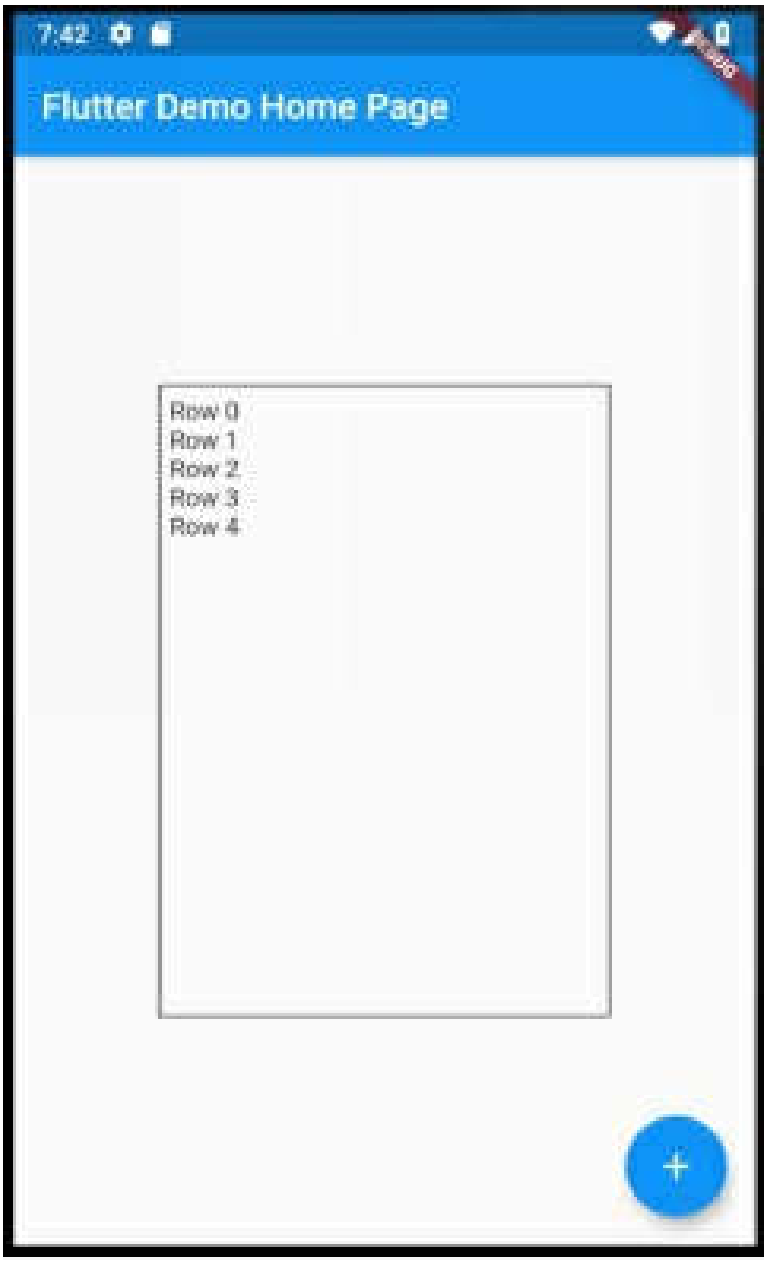
```
        title: new Text("News Feed"),
      ),
      body: new ListView(padding: EdgeInsets.all(20.0),
children: newsCards));
  }
}
```

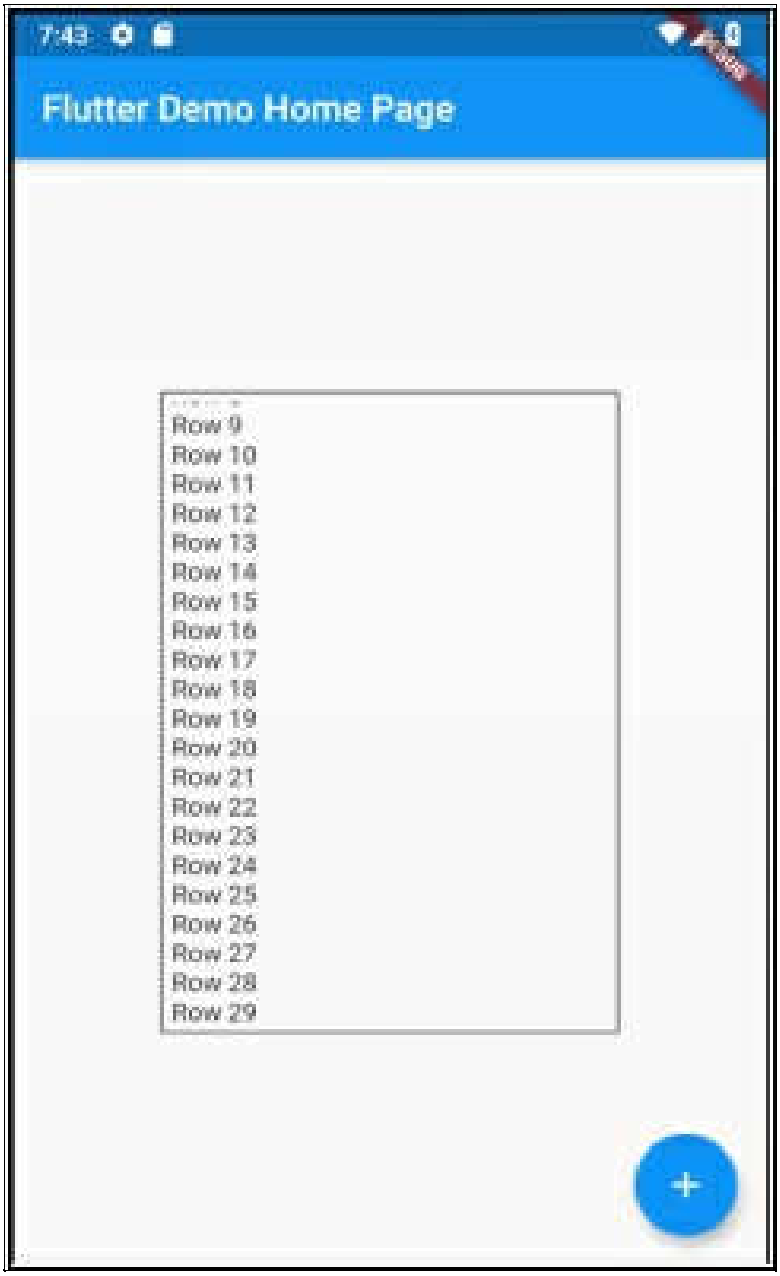
ConstrainedBox

Very useful, you will use this.
A widget that constrains the size of its child widget.

Example – ‘constrained_box’

Here is an example that displays a list of (potentially many) child widgets into a fixed area, with scrolling if necessary. You can on the ‘+’ floating button to add more child widgets. Once you have many child widgets you can drag the list up or down to scroll.


7:42
Flutter Demo Home Page
Row 0
Row 1
Row 2
Row 3
Row 4



Source Code

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

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

class MyApp extends StatelessWidget {
```

```
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
```

```
  }

  @override
  Widget build(BuildContext context) {
    List<Widget> children = [];
    for (int i = 0; i < _counter; i++) {
      children.add(Container(child: Text("Row ${i}")));
    }
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
          child: ConstrainedBox(
              constraints: BoxConstraints(
                maxHeight: 350,
                minHeight: 300,
                minWidth: 200,
                maxWidth: 250,
              ),
              child: Container(
                  decoration: BoxDecoration(border:
Border.all()),
                  padding: EdgeInsets.all(5.0),
                  child: ListView(children: children)))),
      floatingActionButton: FloatingActionButton(
```

```
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

Expanded

A widget that expands a child of a Row, Column, or Flex.

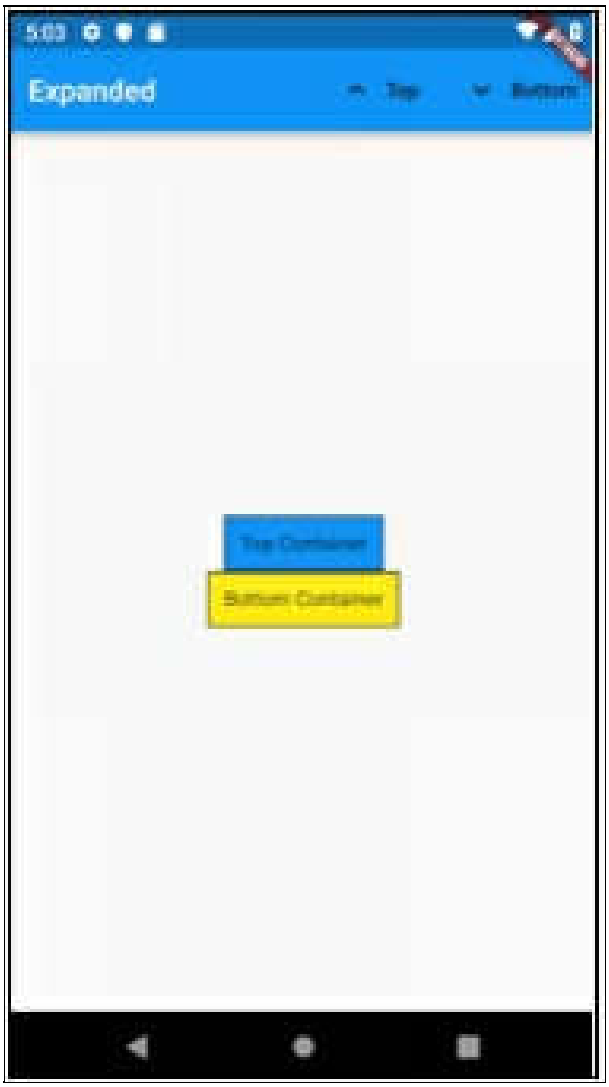
Using an Expanded widget makes a child of a Row, Column, or Flex expand to fill the available space in the main axis (e.g., horizontally for a Row or vertically for a Column). If multiple children are expanded, the available space is divided among them according to the flex factor.

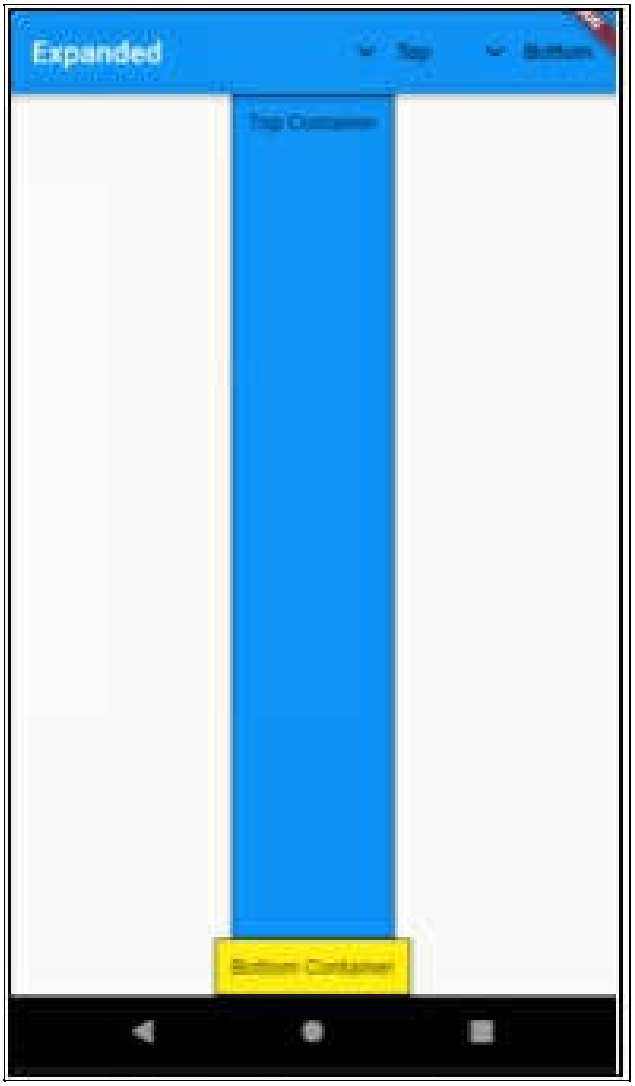
Example – ‘expanded’

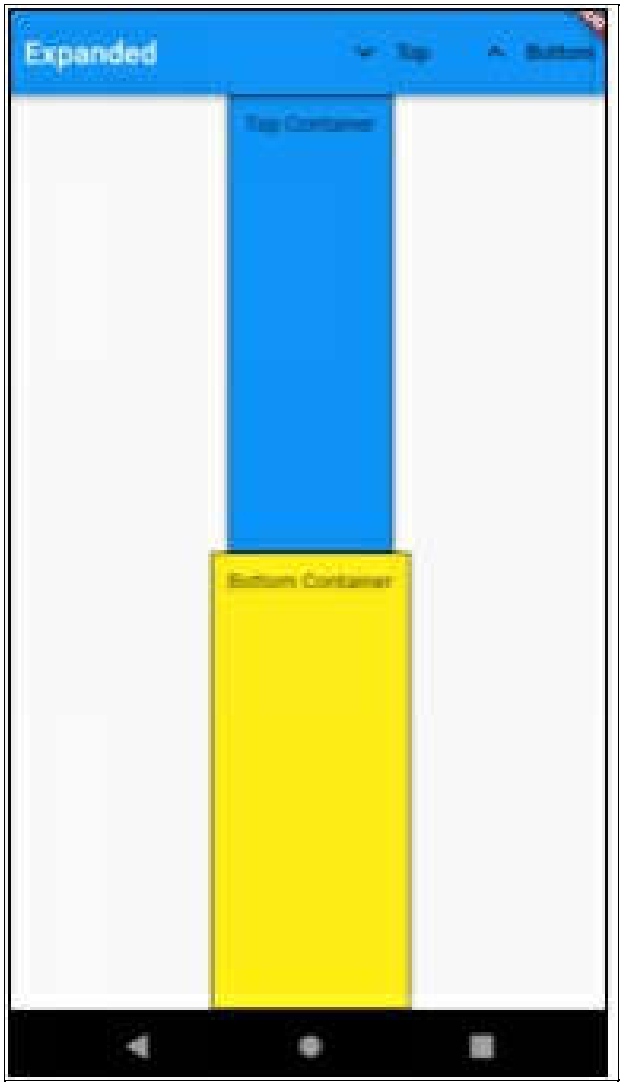
This app shows how two widgets in a column behave when they are contained in a parent Expanded widget (or not).

- If both widgets are expanded, both share the available vertical space evenly.

- If only one is expanded, the expanded one takes up all the available vertical space.
- If neither is expanded, the available vertical space goes unfilled




Expanded
Top
Bottom
Top Container
Bottom Container



Source Code

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
```

```
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  bool _topExpanded = false;
  bool _bottomExpanded = false;

  toggleTop() {
    setState(() => _topExpanded = !_topExpanded);
  }

  toggleBottom() {
    setState(() => _bottomExpanded = !_bottomExpanded);
  }
```

```
  @override
  Widget build(BuildContext context) {
    Container topContainer = Container(
      child: new Text(
        'Top Container',
      ),
      decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 1.0),
          color: Colors.blue),
      padding: EdgeInsets.all(10.0),
    );
    Container bottomContainer = Container(
      child: new Text(
        'Bottom Container',
      ),
      decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 1.0),
          color: Colors.yellow),
      padding: EdgeInsets.all(10.0),
    );
    Widget topWidget =
        _topExpanded ? Expanded(child: topContainer) :
topContainer;
    Widget bottomWidget =
        _bottomExpanded ? Expanded(child: bottomContainer) :
bottomContainer;
    return new Scaffold(
```

```
      appBar: new AppBar(title: new Text("Expanded"),
actions: <Widget>[
        FlatButton.icon(
            icon: Icon(_topExpanded ? Icons.expand_more :
Icons.expand_less),
            label: Text("Top"),
            onPressed: () => toggleTop()),
        FlatButton.icon(
            icon:
                Icon(_bottomExpanded ? Icons.expand_less :
Icons.expand_more),
            label: Text("Bottom"),
            onPressed: () => toggleBottom())
      ]),
      body: new Center(
        child: new Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[topWidget, bottomWidget],
        ),
      ));
  }
}
```

Flexible

This widget is similar to the Expanded widget in that it expands the child Widget, except that it is a little more flexible in regard to Constraints.

When child widgets have Constraints (for example minimum, maximum dimension) then:

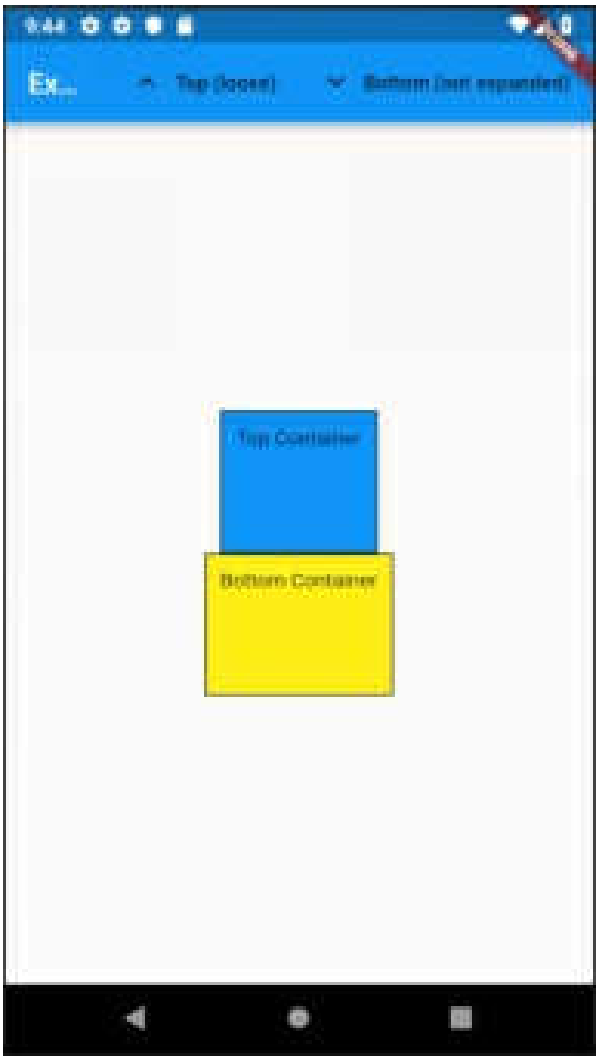
- Expanded Widgets always respect those Constraints, never overriding them.
- Flexible Widgets have the following fit options:
    - Fit ‘expanded’: expands to fit the available screen space, overriding the Constraints.
    - Fit ‘loose’ expands to fit the available screen space, respecting those Constraints, never overriding them.

Example – ‘flexible’

This app shows two Widgets that have a min size of 100 x 100 and a max size of 200 x 200. There are two toolbar buttons to control the use of the available space.

- The top container is expanded / contracted by using a Flexible with a fit that toggles between loose (the child can be at most as large as the available space but is allowed to be smaller). and tight (expands tightly to available space).
- The bottom container is expanded / contacted by

using / not using an Expanded widget.



Source Code

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  bool _topTightFit = false;
  bool _bottomExpanded = false;

  toggleTop() {
    setState(() => _topTightFit = !_topTightFit);
  }

  toggleBottom() {
    setState(() => _bottomExpanded = !_bottomExpanded);
```

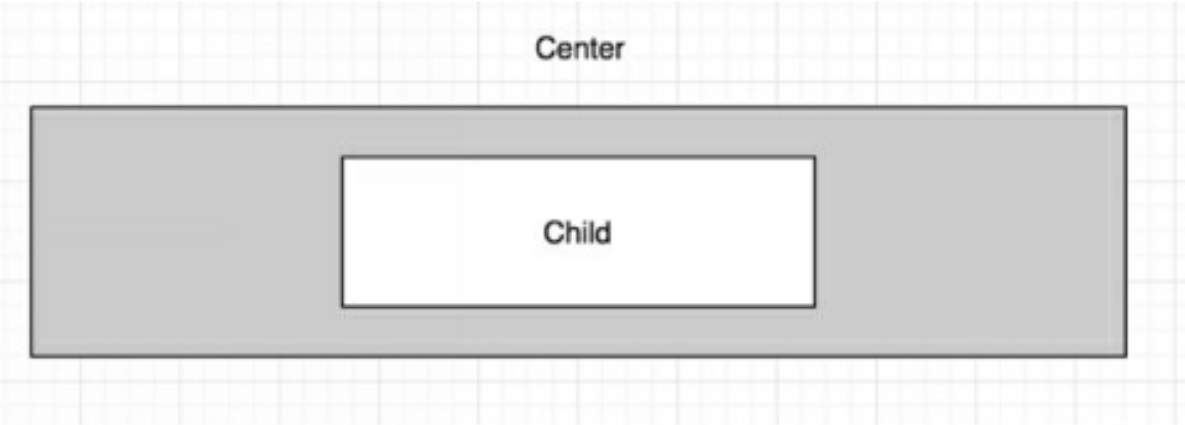
```
  }

  @override
  Widget build(BuildContext context) {
    Container topContainer = Container(
      child: new Text(
        'Top Container',
      ),
      constraints: BoxConstraints(
          minHeight: 100.0, minWidth: 100.0, maxHeight: 200.0,
maxWidth: 200.0),
      decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 1.0),
          color: Colors.blue),
      padding: EdgeInsets.all(10.0),
    );
    Container bottomContainer = Container(
      child: new Text(
        'Bottom Container',
      ),
      constraints: BoxConstraints(
          minHeight: 100.0, minWidth: 100.0, maxHeight: 200.0,
maxWidth: 200.0),
      decoration: BoxDecoration(
          border: Border.all(color: Colors.black, width: 1.0),
          color: Colors.yellow),
      padding: EdgeInsets.all(10.0),
```

```
    );
    Widget topWidget = Flexible(
        child: topContainer, fit: _topTightFit ? FlexFit.tight :
FlexFit.loose);
    Widget bottomWidget =
        _bottomExpanded ? Expanded(child: bottomContainer) :
bottomContainer;
    String toolbarTextTop = "Top (" + (_topTightFit ? "tight" :
"loose") + ")";
    String toolbarTextBottom =
        "Bottom (" + (_bottomExpanded ? "expanded" : "not
expanded") + ")";
    return new Scaffold(
        appBar: new AppBar(title: new Text("Expanded"),
actions: <Widget>[
          FlatButton.icon(
              icon: Icon(_topTightFit
                  ? Icons.keyboard_arrow_up
                  : Icons.keyboard_arrow_up),
              label: Text(toolbarTextTop),
              onPressed: () => toggleTop()),
          FlatButton.icon(
              icon: Icon(_bottomExpanded
                  ? Icons.keyboard_arrow_down
                  : Icons.keyboard_arrow_down),
              label: Text(toolbarTextBottom),
              onPressed: () => toggleBottom())
        ]),
```

```
      body: new Center(
        child: new Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[topWidget, bottomWidget],
        ),
      ));
  }
}
```

Center

This widget is used to center a Widget within its parent Widget.



GestureDetector

A widget that detects gestures.
Often used to add event listeners (like ‘onTop’) onto Widgets that don’t have that capability.

Example – ‘gesture_app’

This app allows the user to try out gestures on a piece of text, logging the recorded gestures in a scrollable textbox below. The user can click on the ‘Clear’ button to clear the textbox.

Source Code

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

```
void main() => runApp(new GestureApp());

class GestureApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Gestures'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  String _log = '';
```

```
  void _clear() {
    setState(() {
      _log = '';
    });
  }

  void _logGesture(String logText) {
    setState(() {
      _log += "\n";
      _log += logText;
    });
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(
          title: new Text(widget.title),
        ),
        body: new Center(
          child: new Column(
            mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
            children: <Widget>[
              GestureDetector(
                  child: Text(
                    'Gesture Me',
```

```
                ),
                onTap: () => _logGesture('tap'),
                onTapDown: (details) =>
_logGesture('onTapDown: ${details}'),
                onTapUp: (details) => _logGesture('onTapUp:
${details}'),
                onTapCancel: () => _logGesture('onTapCancel'),
                onDoubleTap: () => _logGesture('onDoubleTap'),
                onLongPress: () => _logGesture('onLongPress'),
                onVerticalDragDown: (details) =>
                    _logGesture('onVerticalDragDown:
${details}'),
                onVerticalDragStart: (details) =>
                    _logGesture('onVerticalDragStart: ${details}'),
                onVerticalDragUpdate: (details) =>
                    _logGesture('onVerticalDragUpdate'),
                onVerticalDragEnd: (details) =>
                    _logGesture('onVerticalDragEnd: ${details}'),
                onVerticalDragCancel: () =>
                    _logGesture('onVerticalDragCancel'),
                onHorizontalDragDown: (details) =>
                    _logGesture('onHorizontalDragDown:
${details}'),
                onHorizontalDragStart: (details) =>
                    _logGesture('onHorizontalDragStart:
${details}'),
                onHorizontalDragUpdate: (details) =>
                    _logGesture('onHorizontalDragUpdate:
```

```
${details}'),
                onHorizontalDragEnd: (details) =>
                    _logGesture('onHorizontalDragEnd:
${details}'),
                onHorizontalDragCancel: () =>
                    _logGesture('onHorizontalDragCancel')),
            Container(
                child: SingleChildScrollView(child:
Text('$_log')),
                constraints: BoxConstraints(maxHeight: 200.0),
                decoration: BoxDecoration(
                    border: Border.all(
                  color: Colors.grey,
                  width: 1.0,
                )),
                margin: EdgeInsets.all(10.0),
                padding: EdgeInsets.all(10.0)),
            RaisedButton(child: Text('Clear'), onPressed: () =>
_clear())
          ],
        ),
      ));
  }
}
```

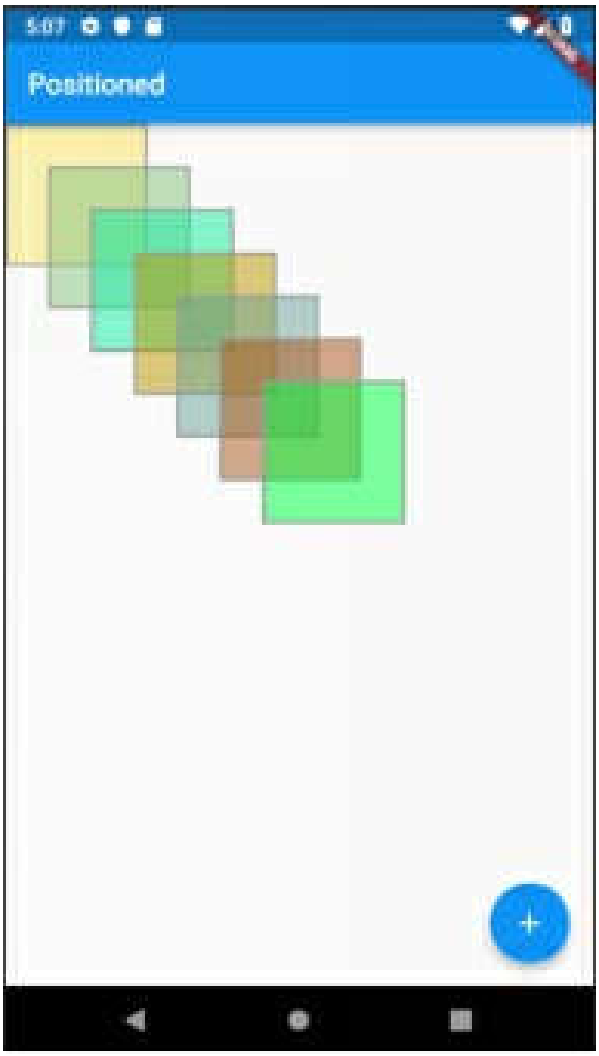
Positioned

Used to wrap a child Widget to control where it is

positioned when added to a group of Widgets stacked using the Stack layout widget.

Example – ‘positioned’

This app allows the user to add another square on top of the existing squares, positioned each time further down and further to the right.



Source Code

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  _MyHomePageState createState() => new
_MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  double _top = 0.0;
```

```
  double _left = 0.0;
  List<Widget> widgetList = [];

  final _random = new Random();

  int next(int min, int max) => min + _random.nextInt(max -
min);

  void _addLayer() {
    setState(() {
      widgetList.add(Positioned(
          left: _left,
          top: _top,
          child: Container(
              width: 100.0,
              height: 100.0,
              decoration: BoxDecoration(
                border: Border.all(
                  color: Colors.grey,
                  width: 2.0,
                ),
                color: Color.fromRGBO(
                    next(0, 255), next(0, 255), next(0, 255), 0.5),
              ))));
    });
    _top += 30;
    _left += 30;
```

```
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text("Positioned"),
      ),
      body: new Stack(children: widgetList),
      floatingActionButton: new FloatingActionButton(
        onPressed: _addLayer,
        tooltip: 'Increment',
        child: new Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

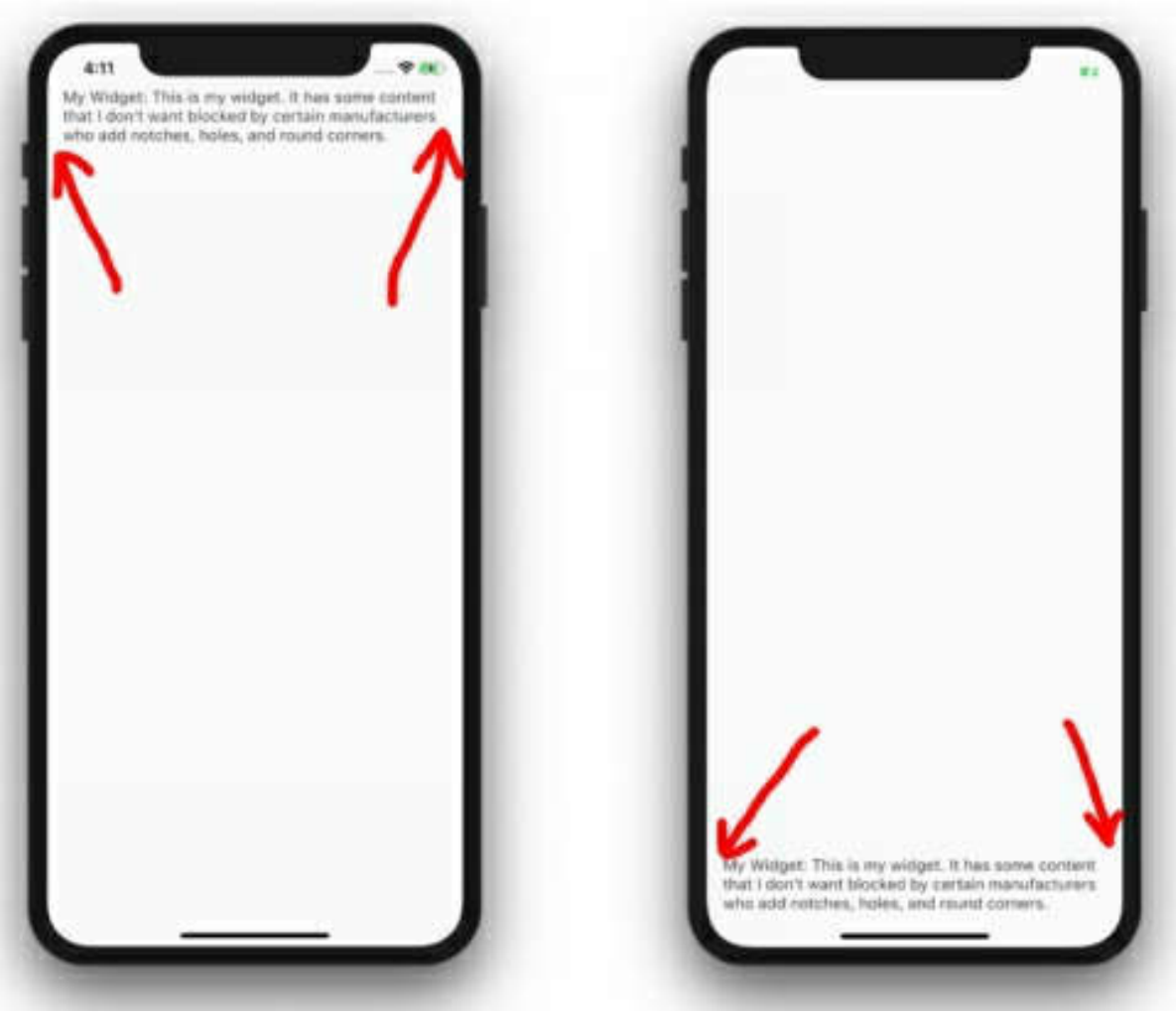
SafeArea

When you wrap a child Widget with a Safe Area, it adds any necessary padding needed to keep your widget from being blocked by the system status bar, notches, holes, rounded corners and other "creative" features by manufactures.

Example: Non-Safe Area

Example: Safe Area

Example: Safe Area with Minimum Padding Set



SingleChildScrollView

This Widget is used to show a child Widget even if there is not enough space to view the entirety of the child Widget.

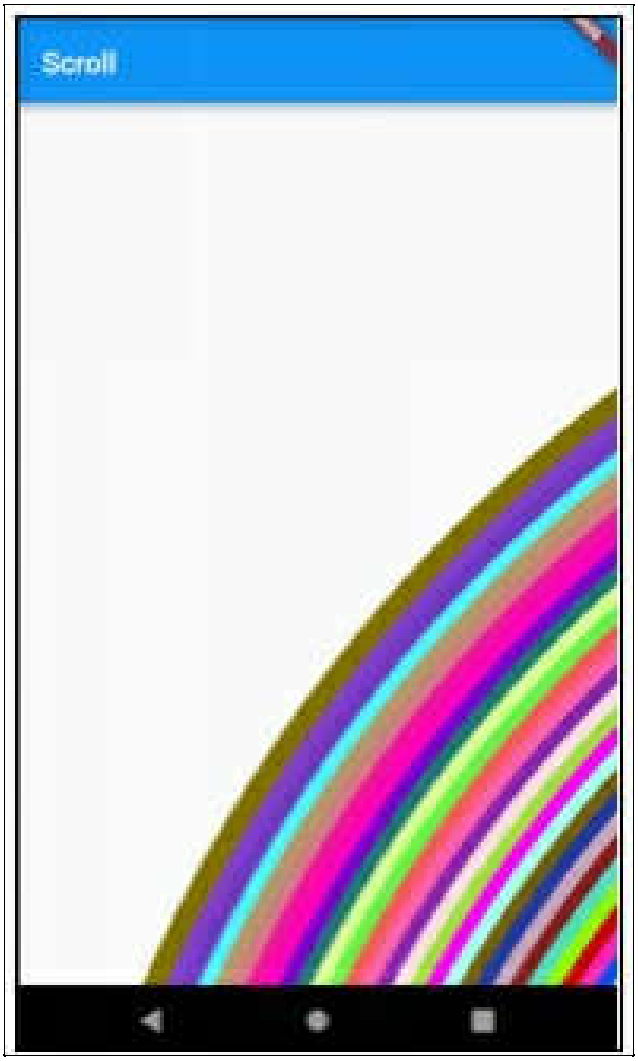
Constructor Arguments Include:

Argument	Description

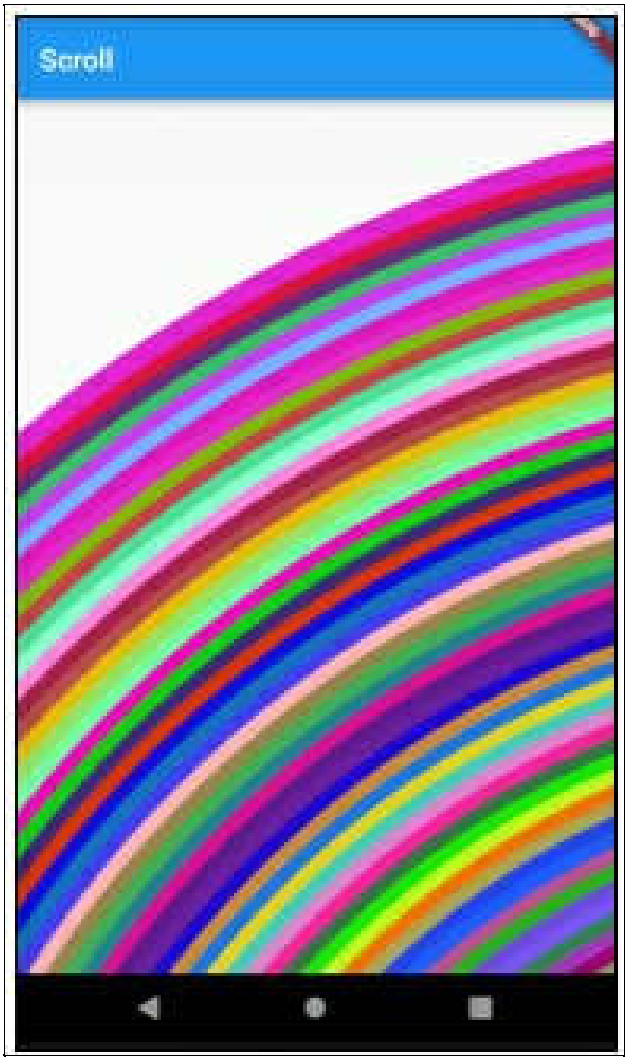
child	Child Widget
scrollDirection	Direction of scrolling. Can be either horizontal or vertical. Cannot be both.
scrollPhysics	How the scroll view continues to animate after the user stops dragging the scroll view.

Exercise – ‘single_child_scroll_view’

This app that displays a very large multicolored globe and allows the user to scroll over it.


Scroll


Scroll



Step 1 – Create Default Flutter App

Follow the instructions in Generate Your First App
Leave project open.

Step 2 – Replace Application Code

Replace contents of file ‘main.dart’ in folder ‘lib’ with the following:

```
import 'dart:math';
```

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new MyHomePage(),
    );
  }
}

class CirclePainter extends CustomPainter {
  final _random = new Random();
  List<Color> _colors = [];

  CirclePainter() {
    for (int i = 0; i < 100; i++) {
      _colors.add(Colors.green
          .withRed(next(0, 255))
          .withGreen(next(0, 255))
          .withBlue(next(0, 255)));
```

```
    }
  }

  int next(int min, int max) => min + _random.nextInt(max -
min);

  @override
  void paint(Canvas canvas, Size size) {
    for (int i = 0; i < 100; i++) {
      var radius = (i * 10).toDouble();
      canvas.drawCircle(
          new Offset(1000.0, 1000.0),
          radius,
          new Paint()
            ..color = _colors[i]
            ..strokeCap = StrokeCap.round
            ..style = PaintingStyle.stroke
            ..strokeWidth = 15.0);
    }
  }

  @override
  bool shouldRepaint(CirclePainter oldDelegate) {
    return false;
  }
}

class MyHomePage extends StatelessWidget {
```

```
  CirclePainter circlePainter = new CirclePainter();
  MyHomePage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(
          title: new Text("Scroll"),
        ),
        body: new SingleChildScrollView(
            scrollDirection: Axis.vertical,
            physics: AlwaysScrollableScrollPhysics(),
            child: CustomPaint(
              size: Size(2000.0, 2000.0),
              foregroundPainter: circlePainter,
            )));
  }
}
```

Step 3 – Open Emulator & Run

Follow the instructions in Open Android Emulator & Run Your First App
You should be able to scroll vertically but not horizontally over the globe.

Step 4 – Change the ‘ScrollDirection’

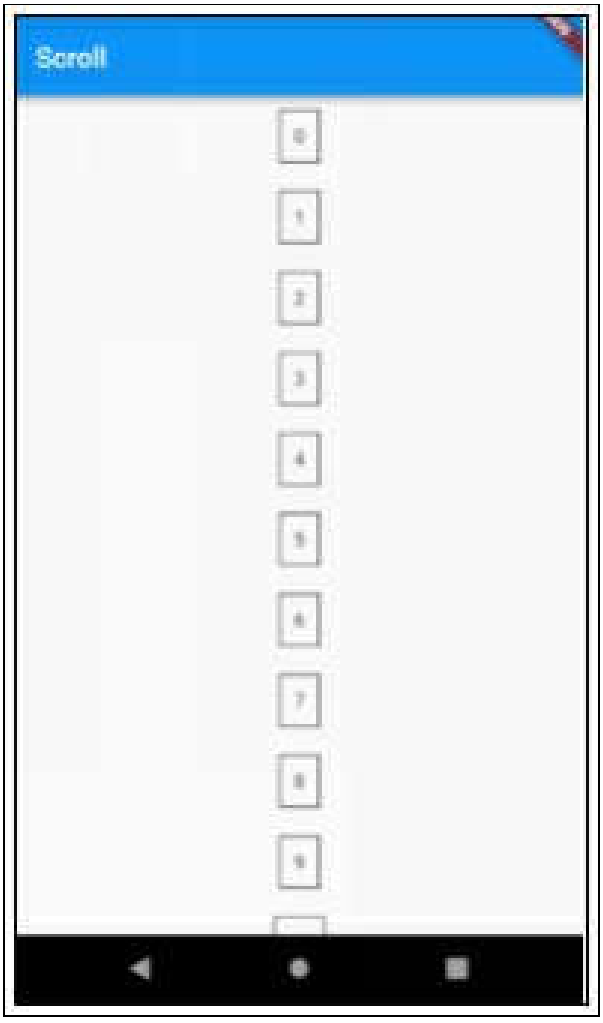
Change the ‘scrollDirection’ constructor Argument of

the SingleChildScrollView from Axis.vertical to Axis.horizontal.

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
      appBar: new AppBar(
        title: new Text("Scroll"),
      ),
      body: new SingleChildScrollView(
          scrollDirection: Axis.horizontal,
          physics: AlwaysScrollableScrollPhysics(),
          child: CustomPaint(
            size: Size(2000.0, 2000.0),
            foregroundPainter: circlePainter,
          )));
}
```

Step 5 – Reload the Changes

You should be able to scroll horizontally but not vertically over the globe.



Step 6 – Edit the ‘build’ Method and Change the SingleChildScrollView to a ListView

```
return new Scaffold(
        appBar: new AppBar(
          title: new Text("Scroll"),
        ),
        body: new Center(
            child: new SingleChildScrollView(
                child: Column(
```

```
    children: childWidgetList,
  ))));
```

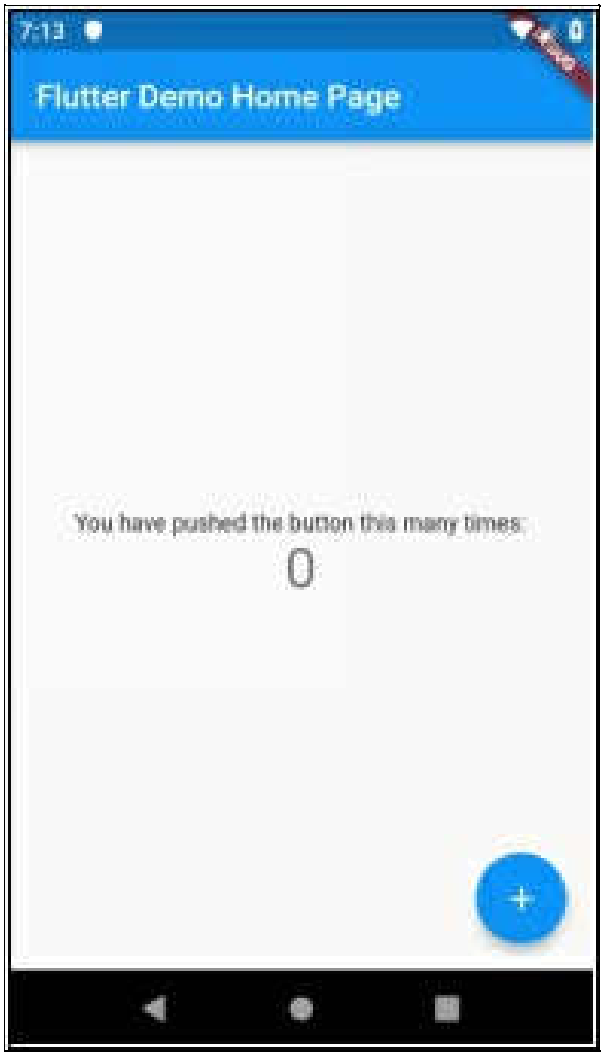
21. App Scaffolding Widgets

Introduction

Flutter makes it easy to generate a default mobile app and you quickly end up with something sophisticated with Color themes, an App Bar, a Content Area with a Count and a Floating Button.

The reason you get something sophisticated so quickly is that the Default App uses Flutter Widgets that were specially designed to scaffold an app as quickly as possible.

The purpose of this chapter is to cover these Widgets.



When your code entry point runs (i.e. the main method), it calls runApp to initialize a given widget (an App Widget). The build method of the App Widget is invoked and it returns a MaterialApp object, which gives Flutter the information it needs to generate the widget and display it on the screen, along with its child Widgets.

So, your App Widget returns a MaterialApp that you

have initialized with the title, theme and home properties initialized. It's called a Material App because this class builds the foundations for an app that uses Google's Material Design UI.

MaterialApp

Builds the foundations for a cross-platform app that uses Google's Material Design UI.
It introduces built-in objects such as the Navigator, Themes and Locales to help you develop your app.

Navigator

We will cover the Navigator in a later chapter.

Themes

When you build a Flutter app, you build a root Widget. That Widget usually returns a MaterialApp, which builds the foundations for the app. One of the constructor arguments for MaterialApp is the Theme object. This object specifies the colors to be used in the application's Widgets. As you can see below the user can pass in Theme data into the MaterialApp constructor using a ThemeData object.

Default Flutter App Uses Blue Theme

```
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new MyHomePage(title: 'Flutter Demo Home Page'),
      debugShowMaterialGrid: true,
      debugShowCheckedModeBanner: false,
      showPerformanceOverlay: true,
    );
  }
}
```

Example of Darkening Theme

Source Code

This is the default Flutter app with just a change to the accent color and the brightness.

```
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
```

```
        accentColor: Colors.redAccent,
        brightness: Brightness.dark),
    home: new MyHomePage(title: 'Flutter Demo Home Page'),
   );
  }
}
```

Locales

In computing, a locale is a set of parameters that defines the user's language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code. The MaterialApp Widget defaults the apps Locale to that of the device it is running on. However, there are locale constructor arguments that let you override the default Locale behavior.

Debugging Constructor Arguments

In addition, the MaterialApp constructor lets you specify additional arguments to enable you to turn on Service Extensions, such as the following:

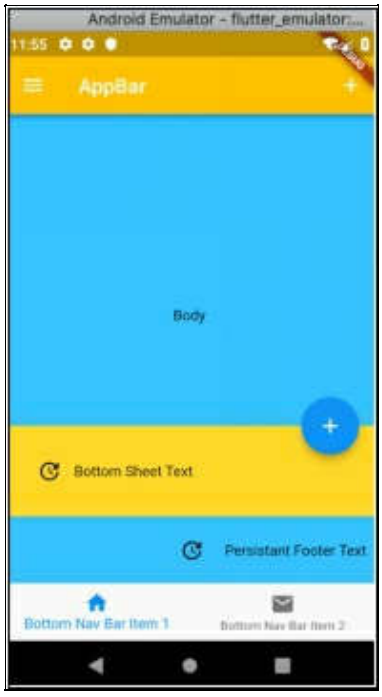
- debugShowMaterialGrid
- showPerformanceOverlay

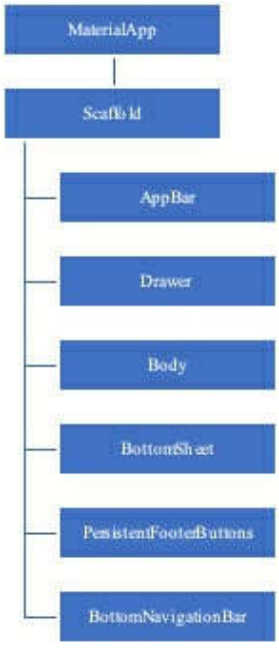
- checkerboardRasterCacheImages
- checkerboardOffscreenLayers
- showSemanticsDebugger
- debugShowCheckedModeBanner

We will cover these later on here: Debugging & Performance Profiling

Scaffold

Provides a pre-determined, standard layout structure for your App.
This Widget serves as the parent Widget of the rest of the Widgets in this chapter.





This will become more obvious when you look the Scaffold Widget exercise in this chapter.

AppBar

App bar with title, icons and menu functionality. Used to display a title plus some icons, which the user can tap on to initiate actions.
You can add an AppBar to your app by specifying the‘appBar’ constructor argument when creating the Scaffold.

Body

Here is where you add the widget that is displayed in the content area of the app.

BottomNavigationBar

Good place to put bottom navigation buttons. The bottom navigation bar is rendered at the bottom, below the Body, BottomSheet and PersistentFooterButtons Widgets. Uses BottomNavigationBarItem items to allow the user to tap on an icon to navigate.

You can add an BottomNavigationBar to your app by specifying the ‘bottomNavigationBar’ constructor

argument when creating the Scaffold.

Drawer

A drawer is an invisible side screen which generally contain menu items and occupies around half of the screen when displayed

You can add a Drawer to the left side of your app by specifying the ‘drawer’ constructor argument when creating the Scaffold. This gives you the Hamburger menu on the AppBar.

You can add a Drawer to the right side your app by specifying the ‘endDrawer’ constructor argument when creating the Scaffold. This does not show a Hamburger menu though.

BottomSheet

Used to show the user information or additional commands without changing the context of what the user is viewing. Used to display content at the bottom of the screen to the user.

Note that there are also ModalBottomSheets that can block the user interface (stop the user from interacting with other content within your application) until the user makes a selection.

You can add a BottomSheet to your app by specifying the ‘bottomSheet’ constructor argument when creating the Scaffold.

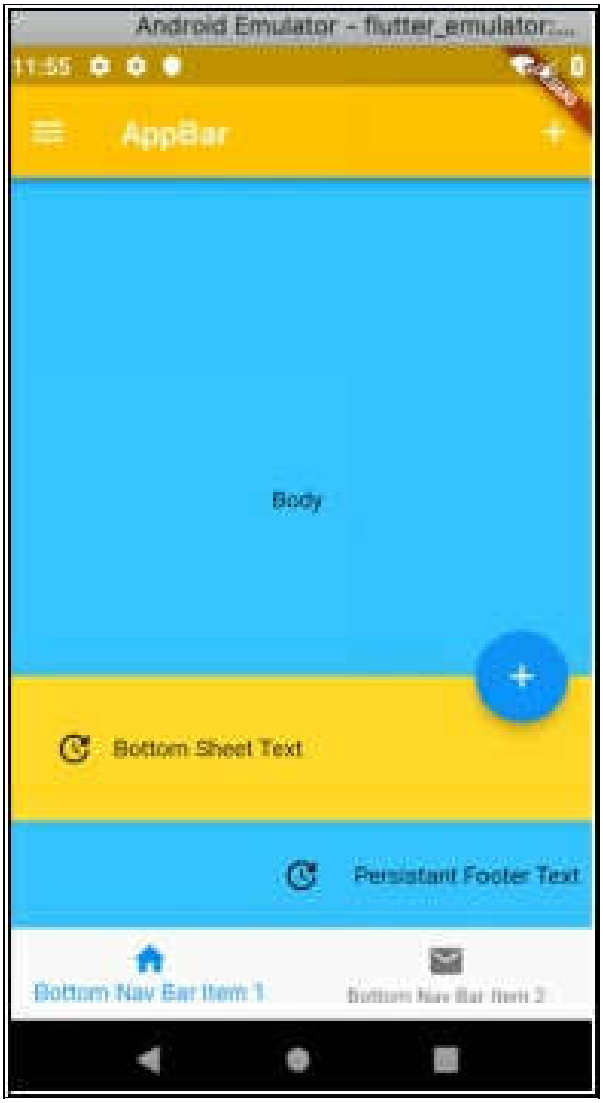
PersistentFooterButtons

Used to show a set of widgets at the bottom of the scaffold above the BottomNavigationBar but below the Body and the BottomSheet. Usually FlatButton widgets. These widgets will be wrapped in a ButtonBar. These buttons are persistently visible, even if the body of the scaffold scrolls.

You can add PersistentFooterButtons to your app by specifying the ‘persistentFooterButtons’ constructor argument when creating the Scaffold.

Exercise – ‘scaffold’

This exercise attempts to use all of the functionality available in the Scaffold Widget.
In doing so it uses all the Widgets that were introduced in this chapter.



Step 1 – Create Default Flutter App

Follow the instructions in [Generate Your First App]
Leave project open.

Step 2 – Replace Application Code

Replace contents of file ‘main.dart’ in folder ‘lib’ with

the following:

```
import ‘package:flutter/material.dart’;

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

class MyApp extends StatelessWidget {
// This widget is the root of your application.
@override
Widget build(BuildContext context) {
  return new MaterialApp(
    title: ‘Flutter Demo’,
    theme: new ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: new MyHomePage(),
  );
}
}

class MyHomePage extends StatelessWidget {
MyHomePage({Key key}) : super(key: key);

@override
Widget build(BuildContext context) {
  return new Scaffold(
      appBar: new AppBar(
        backgroundColor: Colors.amber,
```

```
        title: new Text(“AppBar”),
        actions: <Widget>[
          IconButton(
              icon: Icon(Icons.add),
              onPressed: () {
                print(“Add IconButton Pressed...“);
              })
        ],
      ),
      backgroundColor: Colors.lightBlueAccent,
      body: new Center(
        child: new Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            new Text(
              ‘Body’,
            )
          ],
        ),
      ),
      bottomNavigationBar: BottomNavigationBar(
        type: BottomNavigationBarType.fixed,
        onTap: (index) => debugPrint(“Bottom Navigation Bar
onTap: ${index}“),
        items: [
          BottomNavigationBarItem(
```

```
      icon: new Icon(Icons.home),
      title: new Text('Bottom Nav Bar Item 1'),
    ),
    BottomNavigationBarItem(
      icon: new Icon(Icons.mail),
      title: new Text('Bottom Nav Bar Item 2'),
    )
  ],
),
bottomSheet: Container(
    color: Colors.amberAccent,
    padding: EdgeInsets.all(20.0),
    child: Row(children: <Widget>[
      IconButton(
          icon: Icon(Icons.update),
          onPressed: () {
            print("Bottom Sheet Icon Pressed");
          }),
      Text('Bottom Sheet Text')
    ])),
drawer: Drawer(
    child: ListView(children: <Widget>[
  Row(children: <Widget>[
    IconButton(
        icon: Icon(Icons.add),
        onPressed: () {
```

```
          print(“Drawer Item 1 Pressed”);
        }),
    Text(‘Drawer Item 1’)
  ]),
  Row(children: <Widget>[
    IconButton(
        icon: Icon(Icons.add),
        onPressed: () {
          print(“Drawer Item 2 Pressed”);
        }),
    Text(‘Drawer Item 2’)
  ])
 ])),

 /*

For swiping in from right-side.

endDrawer: Drawer(
   child: ListView(children: <Widget>[
 Row(children: <Widget>[
   IconButton(
      icon: Icon(Icons.add),
      onPressed: () {
        print(“Drawer Item 1");
      }),
   Text(‘Drawer Item 1 Pressed’)
```

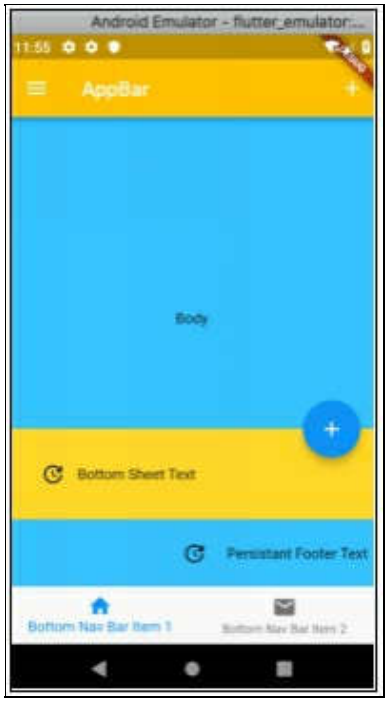
```
      ]),
      Row(children: <Widget>[
        IconButton(
            icon: Icon(Icons.add),
            onPressed: () {
              print("Drawer Item 2 Pressed");
            }),
        Text('Drawer Item 2')
      ])
    ])),
    */
      floatingActionButton: new FloatingActionButton(
          onPressed: () {
            print("FloatingActionButton Pressed");
          },
          tooltip: 'Increment',
          child: new Icon(Icons.add)),
      persistentFooterButtons: <Widget>[
        IconButton(
            icon: Icon(Icons.update),
            onPressed: () {
              print("Persistant Footer Icon Pressed");
            }),
        Text('Persistant Footer Text')
      ]);
}
```

```
}
```

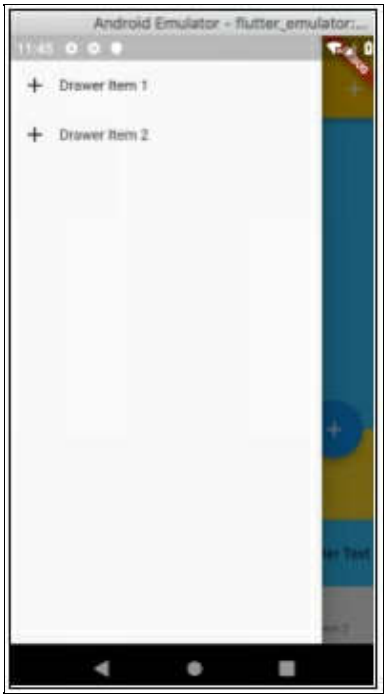
Step 3 – Open Emulator & Run

Follow the instructions in [Open Android Emulator & Run Your First App]

You should get something like the following:



If you touch the hamburger menu on the top left, that opens up the drawer shown to the right.



22. Other Widgets

Introduction

The purpose of this chapter is to cover left-over commonly-used Widgets that have not been covered yet. This does not include Widgets that are used on Forms to enter information. Those will be covered in this chapter: Forms.

Dialog

Dialogs are temporary windows that appear as overlays over the existing application. They are very useful to display something to the user or get user input. When a dialog is displayed, the rest of the app is unavailable. Flutter comes with two boilerplate dialog widgets: Alert Dialog and SimpleDialog. However, you can build custom dialogs quite easily.

AlertDialog

A material design dialog used to display an alert message to the user, with buttons underneath.

To show such a dialog in Flutter, you invoke the ‘showDialog’ method. This method then displays a dialog above the current contents of the app. This

method takes a builder, which in this case returns an instance of the SimpleDialog. This method also returns a [Future] that resolves to the value (if any) that was selected on the dialog. Remember that Futures are covered in the ‘More Advanced Dart’ Chapter.

AlertDialog Constructor Properties

All these properties are optional. However, if you don’t supply anything then nothing will come up!

Name	Description
title	Title.
content	Message or content.
actions	Buttons

Example – ‘alert_dialog’

This app is the same as the default Flutter app, except that it asks you to confirm when you hit the ‘+’ floating button.


Flutter Demo Home Page
Confirm
Are you sure you want to increment the counter?
Yes
No

Source Code

```
import 'dart:async';

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Flutter Demo Home Page'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}
```

```
class _HomeWidgetState extends State<HomeWidget> {
  int _counter = 0;

  Future<bool> _showConfirmDialog() async {
    return await showDialog<bool>(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: const Text('Confirm'),
            content: const Text('Are you sure you want to
increment the '
                'counter?'),
            actions: <Widget>[
              FlatButton(
                onPressed: () {
                  Navigator.pop(context, true);
                },
                child: const Text('Yes'),
              ),
              FlatButton(
                onPressed: () {
                  Navigator.pop(context, false);
                },
                child: const Text('No'),
              )
            ],
```

```
        );
      });
}

void _incrementCounter() {
  _showConfirmDialog().then((result) {
    if (result == true) {
      setState(() {
        _counter++;
      });
    }
  });
}

@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text(widget.title),
    ),
    body: new Center(
      child: new Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          new Text(
            'You have pushed the button this many times:',
          ),
```

```
            new Text(
              '$_counter',
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ),
      floatingActionButton: new FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: new Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for
build methods.
    );
  }
}
```

SimpleDialog

A simple material design dialog used to offer the user a choice between several options. A simple dialog has an optional title that is displayed above the choices.

To show such a dialog in Flutter, you invoke the ‘showDialog’ method. This method then displays a dialog above the current contents of the app. This method takes a builder, which in this case returns an

instance of the SimpleDialog. This method also returns a [Future] that resolves to the value (if any) that was selected on the dialog. Remember that Futures are covered in the ‘More Advanced Dart’ Chapter.

SimpleDialog Constructor Properties

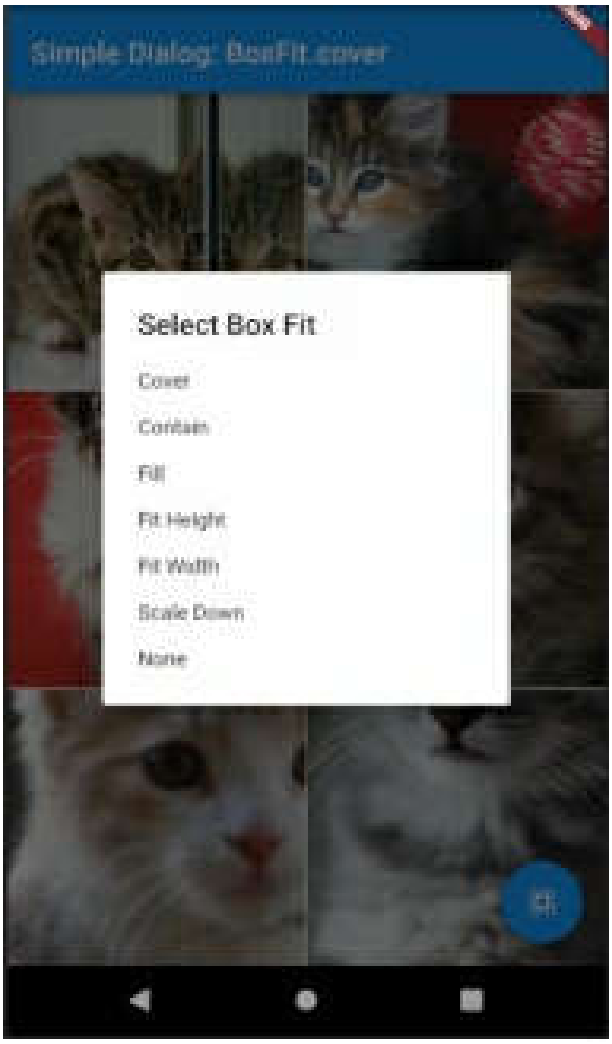
All these properties are optional. However, if you don’t supply anything then nothing will come up!

Name	Description
title	Title.
children	List of Widgets, typically SimpleDialogOptions.

Example – ‘simple_dialog’

This app shows a GridView with kitten images. It allows the user to select how the kitten images are fitted into their available screen space.


Simple Dialog: BoxFit.cover


Simple Dialog: BoxFit.cover
Select Box Fit
Cover
Contain
Fill
Fit Height
Fit Width
Scale Down
None

Source Code

```
import 'dart:async';

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
```

```
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Simple Dialog',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Simple Dialog'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  BoxFit _boxFit = BoxFit.cover;

  void _showBoxFitDialog() async {
    BoxFit boxFit = await showDialog<BoxFit>(
        context: context,
        builder: (BuildContext context) {
          return SimpleDialog(
```

```
title: const Text('Select Box Fit'),
children: <Widget>[
  SimpleDialogOption(
    onPressed: () {
      Navigator.pop(context, BoxFit.cover);
    },
    child: const Text('Cover'),
  ),
  SimpleDialogOption(
    onPressed: () {
      Navigator.pop(context, BoxFit.contain);
    },
    child: const Text('Contain'),
  ),
  SimpleDialogOption(
    onPressed: () {
      Navigator.pop(context, BoxFit.fill);
    },
    child: const Text('Fill'),
  ),
  SimpleDialogOption(
    onPressed: () {
      Navigator.pop(context, BoxFit.fitHeight);
    },
    child: const Text('Fit Height'),
  ),
```

```
          SimpleDialogOption(
            onPressed: () {
              Navigator.pop(context, BoxFit.fitWidth);
            },
            child: const Text('Fit Width'),
          ),
          SimpleDialogOption(
            onPressed: () {
              Navigator.pop(context, BoxFit.scaleDown);
            },
            child: const Text('Scale Down'),
          ),
          SimpleDialogOption(
            onPressed: () {
              Navigator.pop(context, BoxFit.none);
            },
            child: const Text('None'),
          ),
        ],
      );
    });
if (boxFit != null) {
  // not cancelled
  setState(() {
    _boxFit = boxFit;
  });
```

```
    }
  }

  @override
  Widget build(BuildContext context) {
    List<Widget> kittenTiles = [];
    for (int i = 200; i < 1000; i += 100) {
      String imageUrl = "http://placekitten.com/200/${i}";
      kittenTiles.add(GridTile(child: Image.network(imageUrl,
fit: _boxFit)));
    }
    return Scaffold(
      appBar: AppBar(
        title: Text("${widget.title}: ${_boxFit}"),
      ),
      body: OrientationBuilder(builder: (context, orientation) {
        return GridView.count(
            crossAxisCount: (orientation == Orientation.portrait)
? 2 : 3,
            childAspectRatio: 1.0,
            mainAxisSpacing: 1.0,
            crossAxisSpacing: 1.0,
            children: kittenTiles);
      }),
      floatingActionButton: new FloatingActionButton(
        onPressed: _showBoxFitDialog,
        child: new Icon(Icons.select_all),
```

```
      ), // This trailing comma makes auto-formatting nicer for
build methods.
    );
  }
}
```

Custom Dialog Widget

You can build your own Widget and make it visible the ‘showDialog’ method.
Your custom dialog widget will be the child of the boilerplate Dialog Widget:

```
   GridOptions gridOptions = await
showDialog<GridOptions>(
       context: context,
       builder: (BuildContext context) {
         return Dialog(child:
CustomDialogWidget(this._gridOptions));
       });
```

Remember that your code will need to wait for the dialog’s Future to complete in order to get data back from it. Your code in the custom dialog Widget will call Navigator.pop(data) to pass this data back once the it’s closed.

One thing I have noticed from doing custom dialogs in Flutter is that sometimes TextFields do not work well

in them. You tap into a TextField and it flashes the keyboard then it disappears. If this happens then the fix for this is changing:

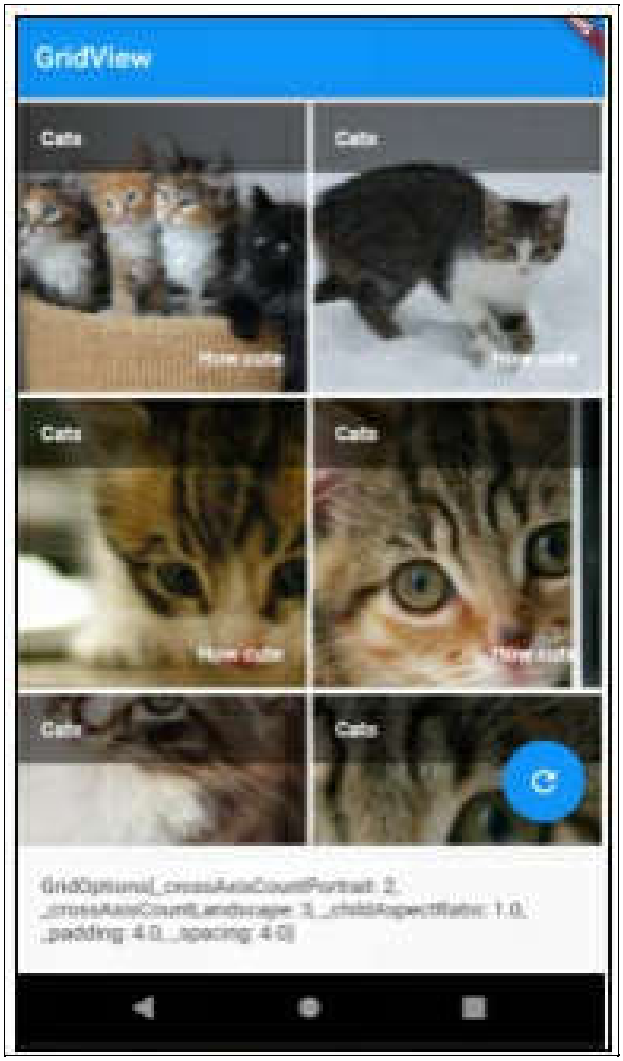
```
final _formKey = GlobalKey<FormState>();
```

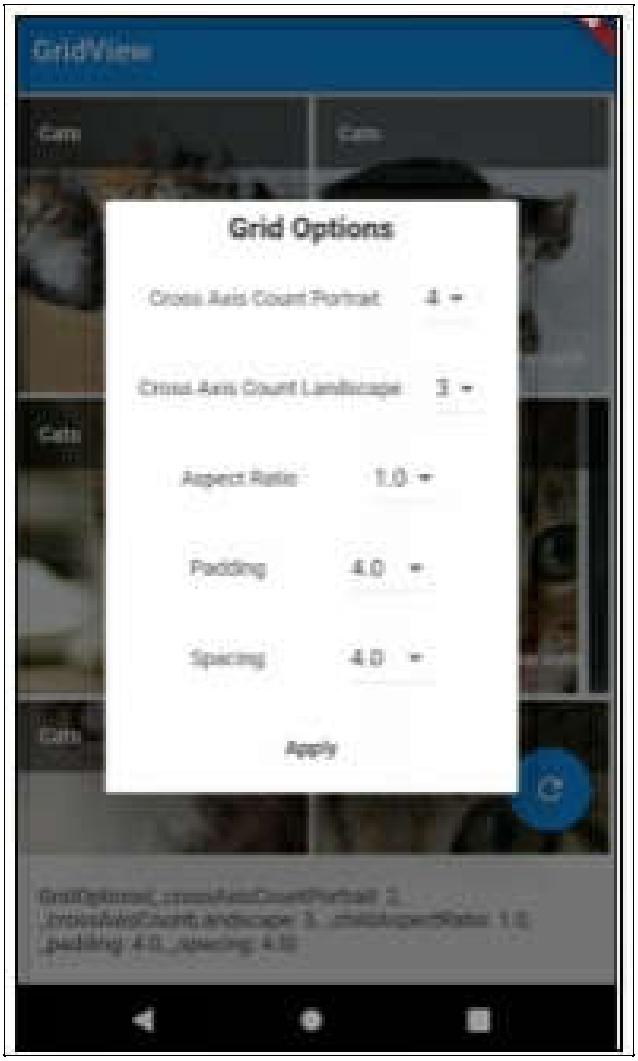
to

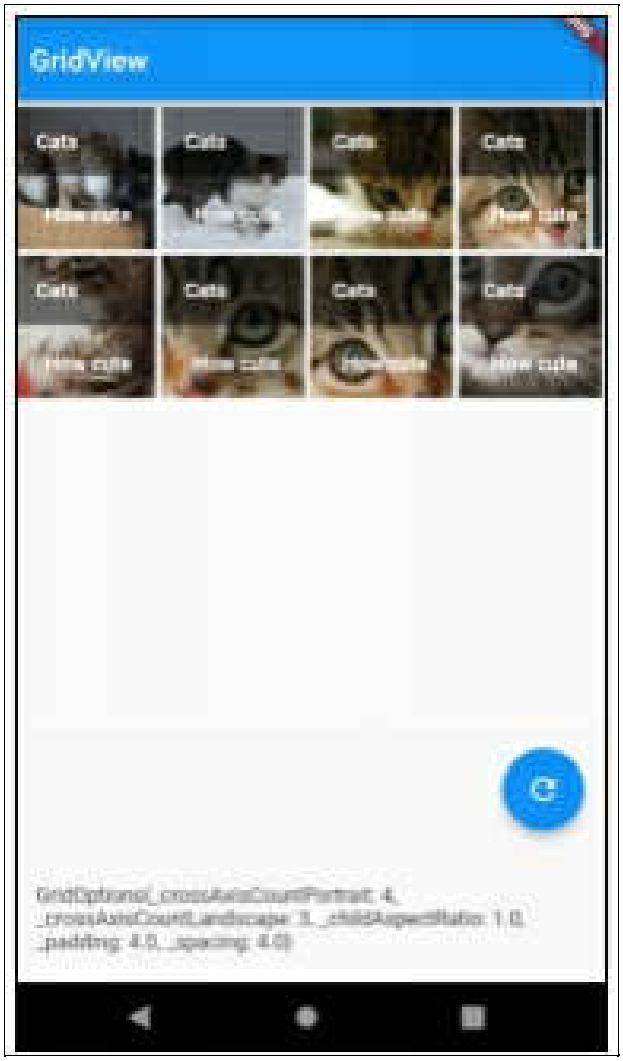
```
static final _formKey = GlobalKey<FormState>();
```

Example – ‘custom_dialog_gridview_settings’

This app shows the grid of cats. It has a button that opens a dialog of the grid options so that the user can change the appearance of the grid.


GridView
Cats
Cats
Cats
Cats
Cats
Cats
GridOptions(_crossAxisCountPortrait: 2, _crossAxisCountLandscape: 3, _childAspectRatio: 1.0, _padding: 4.0, _spacing: 4.0)


GridView
Grid Options
Cross Axis Count Portrait
4
Cross Axis Count Landscape
3
Aspect Ratio
1.0
Padding
4.0
Spacing
4.0
Apply



Source Code

```
import 'dart:async';

import 'package:flutter/material.dart';

void main() => runApp(new GridViewApp());

class GridOptions {
  int _crossAxisCountPortrait;
```

```
  int _crossAxisCountLandscape;
  double _childAspectRatio;
  double _padding;
  double _spacing;

  GridOptions(this._crossAxisCountPortrait,
this._crossAxisCountLandscape,
      this._childAspectRatio, this._padding, this._spacing);

  GridOptions.copyOf(GridOptions gridOptions) {
    this._crossAxisCountPortrait =
gridOptions._crossAxisCountPortrait;
    this._crossAxisCountLandscape =
gridOptions._crossAxisCountLandscape;
    this._childAspectRatio = gridOptions._childAspectRatio;
    this._padding = gridOptions._padding;
    this._spacing = gridOptions._spacing;
  }

  @override
  String toString() {
    return 'GridOptions{_crossAxisCountPortrait:
$_crossAxisCountPortrait, _crossAxisCountLandscape:
$_crossAxisCountLandscape, _childAspectRatio:
$_childAspectRatio, _padding: $_padding, _spacing:
$_spacing}';
  }
}

class GridViewApp extends StatelessWidget {
```

```
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class CustomDialogWidget extends StatefulWidget {
  GridOptions _gridOptions;
  CustomDialogWidget(this._gridOptions) : super();

  @override
  _CustomDialogWidgetState createState() =>
      new
_CustomDialogWidgetState(GridOptions.copyOf(this._gridOpti
}

class _CustomDialogWidgetState extends
State<CustomDialogWidget> {
  GridOptions _gridOptions;

  _CustomDialogWidgetState(this._gridOptions);
```

```
  @override
  Widget build(BuildContext context) {
    return Container(
        height: 400.0,
        width: 250.0,
        child:
            Column(mainAxisAlignment:
MainAxisAlignment.spaceAround, children: <
                Widget>[
          Text("Grid Options",
              style: TextStyle(fontSize: 20.0, fontWeight:
FontWeight.bold)),
          Row(mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
            Spacer(),
            Text("Cross Axis Count Portrait"),
            Spacer(),
            new DropdownButton<int>(
              value: _gridOptions._crossAxisCountPortrait,
              items: <int>[2, 3, 4, 5, 6].map((int value) {
                return new DropdownMenuItem<int>(
                  value: value,
                  child: new Text(value.toString()),
                );
              }).toList(),
              onChanged: (newValue) {
                setState(() {
```

```
              _gridOptions._crossAxisCountPortrait =
newValue;
            });
          },
        ),
        Spacer(),
      ]),
      Row(mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
        Spacer(),
        Text("Cross Axis Count Landscape"),
        Spacer(),
        new DropdownButton<int>(
          value: _gridOptions._crossAxisCountLandscape,
          items: <int>[2, 3, 4, 5, 6].map((int value) {
            return new DropdownMenuItem<int>(
              value: value,
              child: new Text(value.toString()),
            );
          }).toList(),
          onChanged: (newValue) {
            setState(() {
              _gridOptions._crossAxisCountLandscape =
newValue;
            });
          },
        ),
```

```
          Spacer(),
        ]),
        Row(mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[
          Spacer(),
          Text("Aspect Ratio"),
          Spacer(),
          new DropdownButton<double>(
            value: _gridOptions._childAspectRatio,
            items: <double>[1.0, 1.5, 2.0, 2.5].map((double value) {
              return new DropdownMenuItem<double>(
                value: value,
                child: new Text(value.toString()),
              );
            }).toList(),
            onChanged: (newValue) {
              setState(() {
                _gridOptions._childAspectRatio = newValue;
              });
            },
          ),
          Spacer(),
        ]),
        Row(mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[
          Spacer(),
```

```
          Text("Padding"),
          Spacer(),
          new DropdownButton<double>(
            value: _gridOptions._padding,
            items:
                <double>[1.0, 2.0, 4.0, 8.0, 16.0,
32.0].map((double value) {
              return new DropdownMenuItem<double>(
                value: value,
                child: new Text(value.toString()),
              );
            }).toList(),
            onChanged: (newValue) {
              setState(() {
                _gridOptions._padding = newValue;
              });
            },
          ),
          Spacer(),
        ]),
        Row(mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
          Spacer(),
          Text("Spacing"),
          Spacer(),
          new DropdownButton<double>(
            value: _gridOptions._spacing,
```

```
            items:
                <double>[1.0, 2.0, 4.0, 8.0, 16.0,
32.0].map((double value) {
              return new DropdownMenuItem<double>(
                value: value,
                child: new Text(value.toString()),
              );
            }).toList(),
            onChanged: (newValue) {
              setState(() {
                _gridOptions._spacing = newValue;
              });
            },
          ),
          Spacer(),
        ]),
        FlatButton(
            child: Text("Apply"),
            onPressed: () => Navigator.pop(context,
_gridOptions))
      ]));
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key}) : super(key: key);
```

```
  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  List<Widget> _kittenTiles = [];
  GridOptions _gridOptions = GridOptions(2, 3, 1.0, 4.0, 4.0);

  _HomeWidgetState() : super() {
    for (int i = 200; i < 1000; i += 100) {
      String imageUrl = "http://placekitten.com/200/${i}";
      _kittenTiles.add(GridTile(
          header: GridTileBar(
            title: Text("Cats", style: TextStyle(fontWeight:
FontWeight.bold)),
            backgroundColor: Color.fromRGBO(0, 0, 0, 0.5),
          ),
          footer: GridTileBar(
              title: Text("How cute",
                  textAlign: TextAlign.right,
                  style: TextStyle(fontWeight: FontWeight.bold))),
          child: Image.network(imageUrl, fit: BoxFit.cover)));
    }
  }

  void _showGridOptionsDialog() async {
    GridOptions gridOptions = await
```

```
showDialog<GridOptions>(
        context: context,
        builder: (BuildContext context) {
          return Dialog(child:
CustomDialogWidget(this._gridOptions));
        });
    if (gridOptions != null) {
      setState(() {
        _gridOptions = gridOptions;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("GridView"),
      ),
      body: OrientationBuilder(builder: (context, orientation) {
        return GridView.count(
            crossAxisCount: (orientation == Orientation.portrait)
                ? _gridOptions._crossAxisCountPortrait
                : _gridOptions._crossAxisCountLandscape,
            childAspectRatio: _gridOptions._childAspectRatio,
            padding: EdgeInsets.all(_gridOptions._padding),
            mainAxisSpacing: _gridOptions._spacing,
```

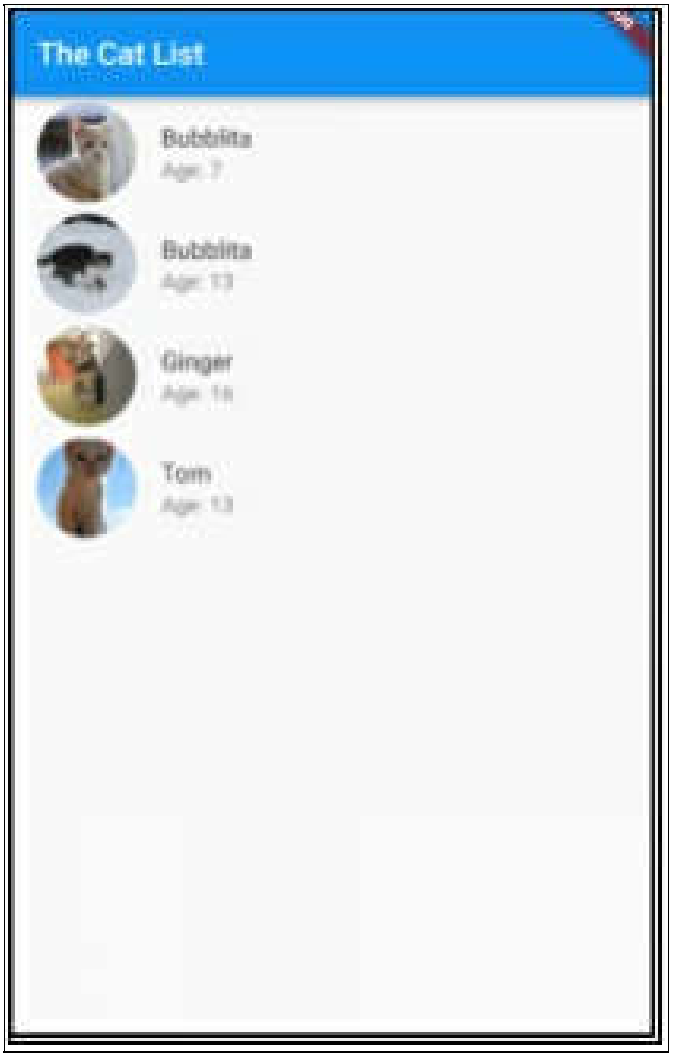
```
            crossAxisSpacing: _gridOptions._spacing,
            children: _kittenTiles);
      }),
      bottomNavigationBar: Container(
         child: Text(_gridOptions.toString()), padding:
EdgeInsets.all(20.0)),
      floatingActionButton: new FloatingActionButton(
        onPressed: _showGridOptionsDialog,
        tooltip: 'Try more grid options',
        child: new Icon(Icons.refresh),
      ), // This trailing comma makes auto-formatting nicer for
build methods.
    );
  }
}
```

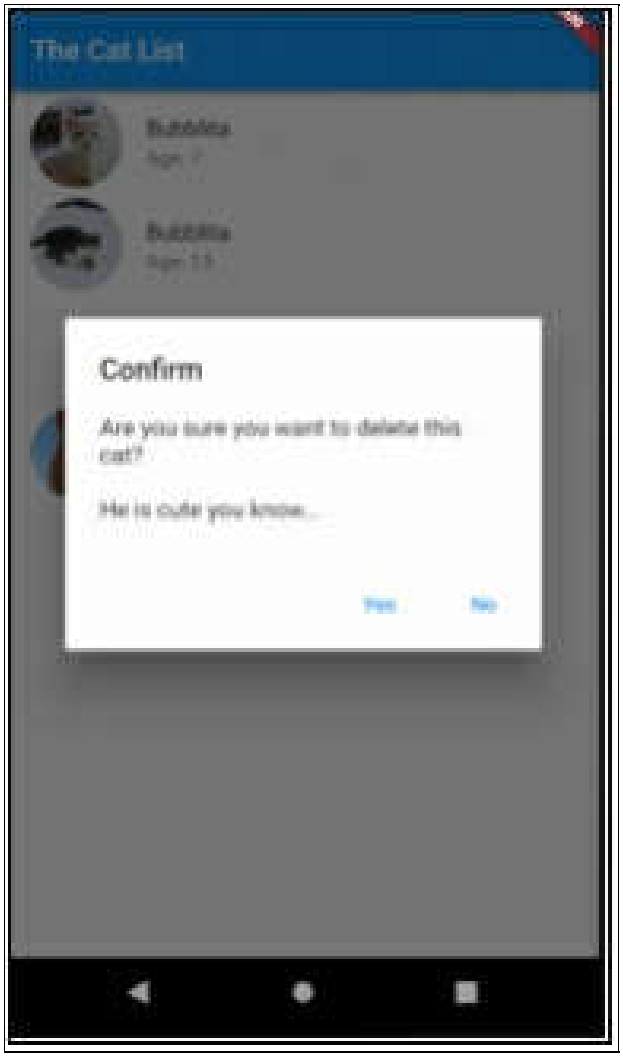
Dismissible

This Widget is useful if you want to be able to swipe left on lists to delete items.
You can also specify other swiping directions in the constructor.

Example – ‘dismissible’

This app shows a list of cats. You can swipe left on a cat to delete him/her.


The Cat List
Bubbilta
Age: 7
Bubbilta
Age: 13
Ginger
Age: 16
Tom
Age: 13



Source Code

```
import 'dart:math';

import 'package:flutter/material.dart';

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

class Cat {
  String imageSrc;
```

```
  String name;
  int age;
  int votes;

  Cat(this.imageSrc, this.name, this.age, this.votes);

  operator ==(other) => (other is Cat) && (imageSrc ==
other.imageSrc);

  int get hashCode => imageSrc.hashCode;
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Cat List',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'The Cat List'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
```

```
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final GlobalKey<AnimatedListState> _listKey =
GlobalKey();
  List<String> CAT_NAMES = [
    "Tom",
    "Oliver",
    "Ginger",
    "Pontouf",
    "Madison",
    "Bubblita",
    "Bubbles"
  ];

  Random _random = Random();
  List<Cat> _cats = [];

  int next(int min, int max) => min + _random.nextInt(max -
min);

  _MyHomePageState() : super() {
    for (int i = 200; i < 250; i += 10) {
      _cats.add(Cat("http://placekitten.com/200/${i}",
CAT_NAMES[next(0, 6)],
```

```
        next(1, 32), 0));
    }
  }

  _buildItem(Cat cat, {int index = -1}) {
    return ListTile(
        key: Key("ListTile:${cat.hashCode.toString()}"),
        leading: CircleAvatar(
            backgroundImage: NetworkImage(cat.imageSrc),
radius: 32.0),
        title: Text(cat.name, style: TextStyle(fontSize: 25.0)),
        subtitle: Text("This little thug is ${cat.age} year(s)
old.",
            style: TextStyle(fontSize: 15.0)));
  }

  _onDismissed(int index) {
    // If you do the code below
    // setState(() {
    //   _cats.remove(index);
    // });
    // then you get the following error:
    // This MyHomePage widget cannot be marked as needing
to build because the framework is already in the
    // process of building widgets. A widget can be marked as
needing to be built during the build phase
    // only if one of its ancestors is currently building. This
exception is allowed because the framework
```

```
    // builds parent widgets before children, which means a
dirty descendant will always be built.
    // Otherwise, the framework might not visit this widget
during this build phase.
    //
    // This works:
    _cats.remove(index);

  }

  Future<bool> _confirmDismiss(DismissDirection direction)
async{
    return await showDialog<bool>(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: const Text('Confirm'),
            content: Text('Are you sure you want to delete this
cat?\n\nHe is cute you know...'),
            actions: <Widget>[
              FlatButton(
                onPressed: () {
                  Navigator.pop(context, true);
                },
                child: const Text('Yes'),
              ),
              FlatButton(
                onPressed: () {
```

```
              Navigator.pop(context, false);
            },
            child: const Text('No'),
          )
        ],
      );
    });
}

@override
Widget build(BuildContext context) {
 return Scaffold(
    appBar: AppBar(
     title: Text(widget.title),
    ),
    body: ListView.builder(
       itemCount: _cats != null ? _cats.length : 0,
       itemBuilder: (context, index) {
         Cat cat = _cats[index];
         return Dismissible(
            confirmDismiss: _confirmDismiss,
            direction: DismissDirection.endToStart,
            onDismissed: _onDismissed(index),
            key: ValueKey(cat.hashCode.toString()),
            child: ListTile(
               leading: CircleAvatar(
```

```
                    backgroundImage:
NetworkImage(cat.imageSrc),
                    radius: 32.0),
                title: Text('${cat.name}'),
                subtitle: Text('Age: ${cat.age}')));
        }));
  }
}
```

ExpansionPanelList & ExpansionPanel

These two widgets are designed to work together to present a list of expandable panels to the user. They help you build a UI with expanding lists but they **don’t** hold the state for you.

You have to manage the state of what was expanded / collapsed and rebuild the ExpansionPanelList & ExpansionPanels everytime the state changes. This sounds slow but it’s not! Check out the example below to see an example of this.

ExpansionPanelList

This does the following:

- Lays out the child ExpansionPanels.
- Provides expansionCallback constructor

argument to which you can add provide to respond to the user attempting to expand / collapse panels, managing the state and forcing a repaint once a panel is expanded or collapsed.
- Animations.

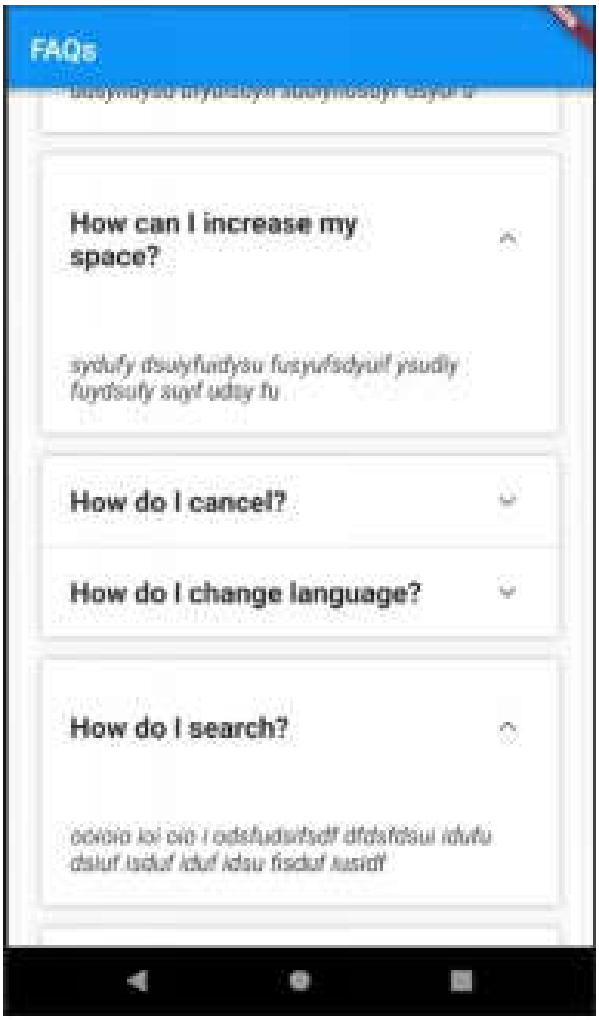
ExpansionPanel

This does the following:

- Display the header with an arrow next to it.
- Displays the body if the ‘isExpanded’ constructor argument is set to true.
- When the user clicks on header arrow to expand or collapse, this fires the expansionCallback in the ExpansionPanelList.

Example – ‘expansion_panel’

This app shows Frequently Asked Questions with arrows. When the user taps the arrow on a question, the panel is expanded to show the answer.



Source Code

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

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

class ExpansionPanelData {
  String _title;
  String _body;
  bool _expanded;
```

```
  ExpansionPanelData(this._title, this._body, this._expanded);

  String get title => _title;

  @override
  String toString() {
    return 'ExpansionPanelData{_title: $_title, _body: $_body,
_expanded: $_expanded}';
  }

  String get body => _body;

  bool get expanded => _expanded;

  set expanded(bool value) {
    _expanded = value;
  }
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
        title: 'Flutter Demo',
        theme: new ThemeData(
          primarySwatch: Colors.green,
        ),
        home: new HomeWidget(),
```

```
        showPerformanceOverlay: true);
  }
}

class HomeWidget extends StatefulWidget {
  @override
  _HomeWidgetState createState() => new
_HomeWidgetState([
        ExpansionPanelData(
            "Can I backup my data?",
            "dsfuysdiu fudsy fiusdyf "
            "usdyf iudsyf udsyfiuysd ufyuisdyfi sduiyfiusdyf
dsyui u",
            false),
        ExpansionPanelData(
            "How can I increase my space?",
            "sydufy "
            "dsuiyfuidysu fusyufsdyuif ysudiy fuydsufy suyf udsy
fu",
            false),
        ExpansionPanelData(
            "How do I cancel?",
            "ddsufysd yfds fsduyf sdyf "
            "sudyuy fsudyf sydyf dsy fdsuyf udsufy
udsyfdsfyuysdf uyud",
            false),
        ExpansionPanelData(
```

```
            "How do I change language?",
            "udsuf sdifuu fdsuif "
            "uf dsufdisu fius wewqw qeqweqwyiquuiqweqwewqe
weewe wewe",
            false),
        ExpansionPanelData(
            "How do I search?",
            "ooioio ioi oio i odsfudsifsdf"
            " dfdsfdsui idufu dsiuf isduf iduf idsu fisduf iusidf ",
            false),
        ExpansionPanelData(
            "How do I view on other devices?",
            "idusdf isu "
            "idsu idsu fisduf usyfuedy ewuyduyed uyeu dyeudy
uweyu",
            false),
        ExpansionPanelData(
            "How do I view my history",
            "iirewy syfudy fu "
            "yfsduyfds yfdsuyf udsfydsufy sduyf dsuyf udsyf
udsyuee",
            false),
        ExpansionPanelData(
            "Is my subscription cost going to go up?",
            "wieureiy dys udsyyf "
            "dsufy dusyfudsy fuysdu
udsyuyfudsyfuewyrwreooioou  uiy",
```

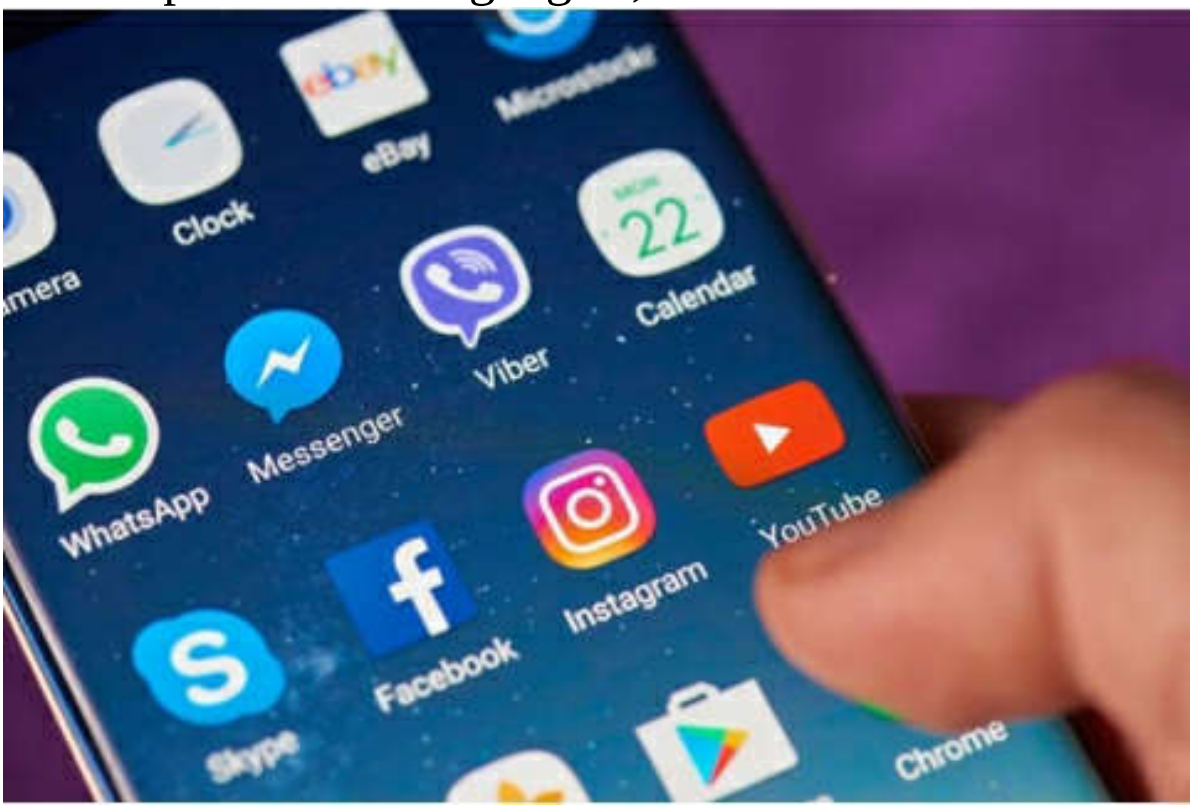
```
          false),
      ]);
}

class _HomeWidgetState extends State<HomeWidget> {
  // Track expansion panels, including expanded true/false;
  List<ExpansionPanelData> _expansionPanelData;

  _HomeWidgetState(this._expansionPanelData);

  _onExpansion(int panelIndex, bool isExpanded) {
    // Toggle the expanded state. Using setState will force
'build' to fire.
    setState(() {
      _expansionPanelData[panelIndex].expanded =
          !(_expansionPanelData[panelIndex].expanded);
    });
  }

  @override
  Widget build(BuildContext context) {
    // Build the expansion panels from scratch every time the ui
builds.
    // This is not as expensive as it sounds.
    List<ExpansionPanel> expansionPanels = [];
    for (int i = 0, ii = _expansionPanelData.length; i < ii; i++)
{
      var expansionPanelData = _expansionPanelData[i];
```

```
    expansionPanels.add(ExpansionPanel(
        headerBuilder: (BuildContext context, bool
isExpanded) {
          return Padding(
              padding: EdgeInsets.all(20.0),
              child: Text(expansionPanelData.title,
                  style: TextStyle(
                      fontSize: 20.0, fontWeight:
FontWeight.bold)));
        },
        body: Padding(
            padding: EdgeInsets.all(20.0),
            child: Text(expansionPanelData.body,
                style:
                    TextStyle(fontSize: 16.0, fontStyle:
FontStyle.italic))),
        isExpanded: expansionPanelData.expanded));
  }
  return new Scaffold(
      appBar: new AppBar(
        title: new Text("FAQs"),
      ),
      body: SingleChildScrollView(
          child: Container(
        margin: const EdgeInsets.all(24.0),
        child: new ExpansionPanelList(
            children: expansionPanels, expansionCallback:
```

```
_onExpansion),
        )));
  }
}
```

GridView

Grids are very commonly-used on devices to present many items of information in a small screen area in a clear manner. Typically, your launch (or home) screen will be presented using a grid, see below.



Notice how each Grid item is of a uniform size (unlike a staggered grid, see below).
Grids are often fluid – users can view a certain number

of items per grid row in portrait mode and a different number of items per grid row in landscape mode.

The Flutter GridView Widget enables developers to quickly build grids. The GridView Widget is very flexible and here are some of the more popular options that are available as properties in the constructor:

- crossAxisCount – number of items per grid row
- childAspectRatio – sets the aspect ratio of each item in the grid
- padding – padding around the grid
- mainAxisSpacing – spacing between items in the grid on main axix
- crossAxisSpacing – spacing between items in the grid on cross axis
- children – array of child widgets to be displayed as items

Builder

The GridView has a builder to improve the performance of the Grid when you have to display many items. This is covered in the Builder chapter.

GridTile

You don’t have to use GridTiles with GridViews but

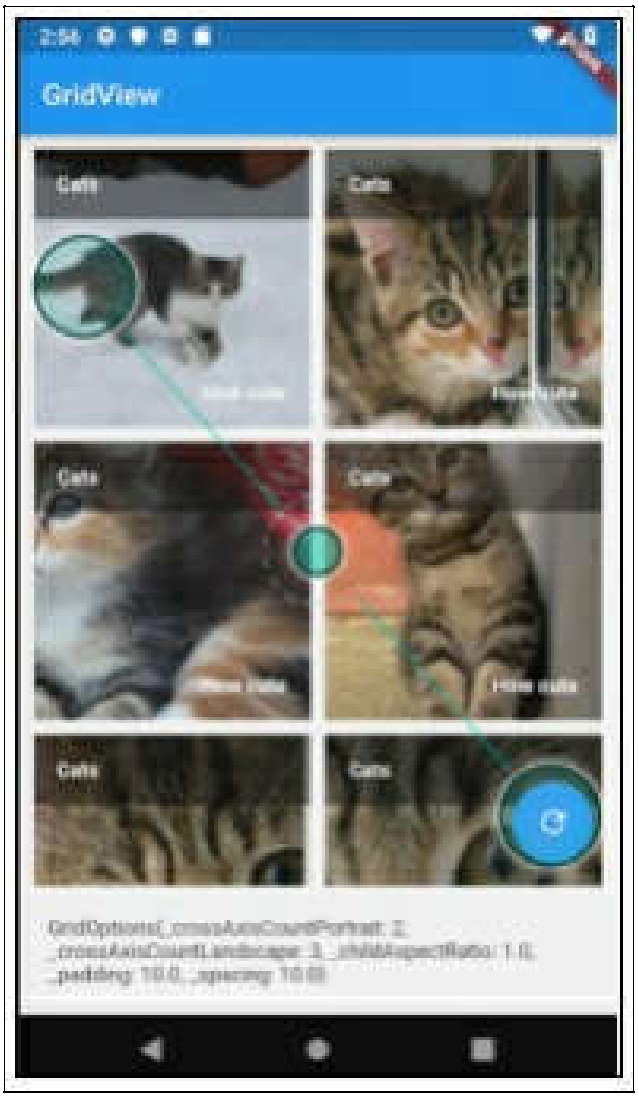
they are useful because they can display headers and footers (using GridTileBars) for each item. Really useful when you want to add some text, description or price to each item.

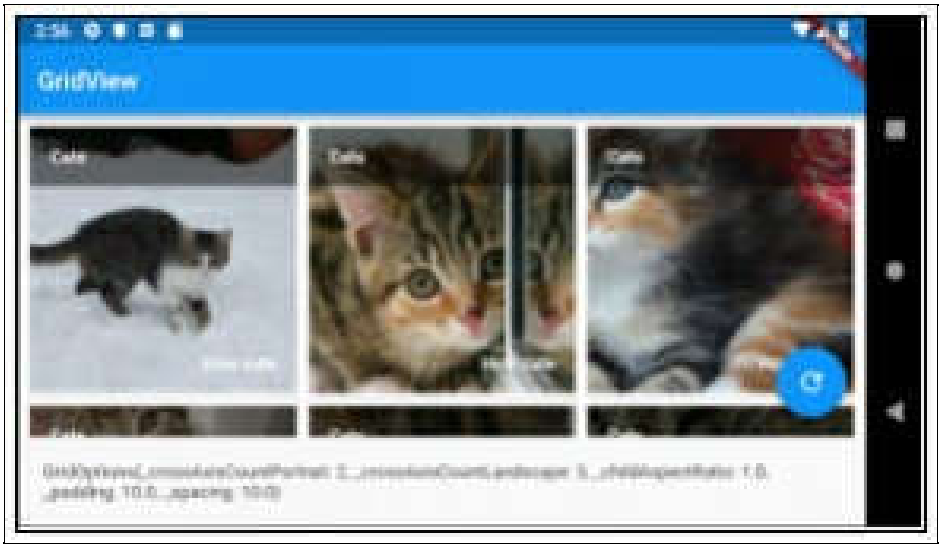
GridTileBar

Used to show headers or footers on grid tiles.

Example – ‘gridview_app’

This is an app that shows kittens on a grid. It has a refresh button that enables you to cycle through some example grid options and see how they affect the appearance of the grid. Also note that the grid always works responsively, changing the number of items per grid row when the device changes from portrait to landscape and visa-versa. This app also uses GridTile and Grid TileBar widgets.


GridView
Cats
Cats
Cats
Cats
Cats
Cats
GridOptions(_crossAxisCountPortrait: 2,
_crossAxisCountLandscape: 3, _childAspectRatio: 1.0,
_padding: 10.0, _spacing: 10.0)


GridView
Cats
Cats
Cats

Source Code

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

void main() => runApp(new GridViewApp());

class GridOptions {
  int _crossAxisCountPortrait;
  int _crossAxisCountLandscape;
  double _childAspectRatio;
  double _padding;
  double _spacing;

  GridOptions(this._crossAxisCountPortrait,
this._crossAxisCountLandscape,
      this._childAspectRatio, this._padding, this._spacing);

  @override
  String toString() {
    return 'GridOptions{_crossAxisCountPortrait:
$_crossAxisCountPortrait, _crossAxisCountLandscape:
$_crossAxisCountLandscape, _childAspectRatio:
$_childAspectRatio, _padding: $_padding, _spacing:
$_spacing}';
  }
}

class GridViewApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
```

```
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Flutter Demo Home Page'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
}

class _HomeWidgetState extends State<HomeWidget> {
  List<Widget> _kittenTiles = [];
  int _gridOptionsIndex = 0;
  List<GridOptions> _gridOptions = [
    GridOptions(2, 3, 1.0, 10.0, 10.0),
    GridOptions(3, 4, 1.0, 10.0, 10.0),
    GridOptions(4, 5, 1.0, 10.0, 10.0),
```

```
    GridOptions(2, 3, 1.0, 10.0, 10.0),
    GridOptions(2, 3, 1.5, 10.0, 10.0),
    GridOptions(2, 3, 2.0, 10.0, 10.0),
    GridOptions(2, 3, 1.0, 10.0, 10.0),
    GridOptions(2, 3, 1.5, 20.0, 10.0),
    GridOptions(2, 3, 2.0, 30.0, 10.0),
    GridOptions(2, 3, 1.0, 10.0, 10.0),
    GridOptions(2, 3, 1.5, 10.0, 20.0),
    GridOptions(2, 3, 2.0, 10.0, 30.0),
  ];

  _HomeWidgetState() : super() {
    for (int i = 200; i < 1000; i += 100) {
      String imageUrl = "http://placekitten.com/200/${i}";
      _kittenTiles.add(GridTile(
          header: GridTileBar(
            title: Text("Cats", style: TextStyle(fontWeight:
FontWeight.bold)),
            backgroundColor: Color.fromRGBO(0, 0, 0, 0.5),
          ),
          footer: GridTileBar(
              title: Text("How cute",
                  textAlign: TextAlign.right,
                  style: TextStyle(fontWeight: FontWeight.bold))),
          child: Image.network(imageUrl, fit: BoxFit.cover)));
    }
  }
```

```
void _tryMoreGridOptions() {
  setState(() {
    _gridOptionsIndex++;
    if (_gridOptionsIndex >= (_gridOptions.length - 1)) {
      _gridOptionsIndex = 0;
    }
  });
}

@override
Widget build(BuildContext context) {
  GridOptions options = _gridOptions[_gridOptionsIndex];
  return Scaffold(
    appBar: AppBar(
      title: Text("GridView"),
    ),
    body: OrientationBuilder(builder: (context, orientation) {
      return GridView.count(
          crossAxisCount: (orientation == Orientation.portrait)
              ? options._crossAxisCountPortrait
              : options._crossAxisCountLandscape,
          childAspectRatio: options._childAspectRatio,
          padding: EdgeInsets.all(options._padding),
          mainAxisSpacing: options._spacing,
          crossAxisSpacing: options._spacing,
          children: _kittenTiles);
```

```
      }),
      bottomNavigationBar: Container(
         child: Text(options.toString()), padding:
EdgeInsets.all(20.0)),
      floatingActionButton: new FloatingActionButton(
        onPressed: _tryMoreGridOptions,
        tooltip: 'Try more grid options',
        child: new Icon(Icons.refresh),
      ), // This trailing comma makes auto-formatting nicer for
build methods.
    );
  }
}
```

Further Reading

- This is an excellent article about writing staggered gridviews. These are excellent at displaying items of different sizes.

https://medium.com/@lets4r/flutorial-create-a-staggered-gridview-9c881a9b0b98

PopupMenuButton

Displays a menu when pressed and calls ‘onSelected’ when the menu is dismissed because an item was selected. The value passed to ‘onSelected’ is the value

of the selected menu item.

Example – ‘popup_menu_button’

This app is similar to the default Flutter app except that it enables the user to increment the counter using the menu. The menu also has an exit option to close the app.

Source Code

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

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

enum PopupMenuAction { add1, add10, add100, exit }

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(title: 'Flutter Demo Home Page'),
    );
  }
}

class HomeWidget extends StatefulWidget {
  HomeWidget({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _HomeWidgetState createState() => new
_HomeWidgetState();
```

```
}

class _HomeWidgetState extends State<HomeWidget> {
  int _counter = 0;

  void _increment(int by) {
    setState(() {
      _counter += by;
    });
  }

  void _onPopupMenuSelected(PopupMenuAction item) {
    if (PopupMenuAction.exit == item) {
      SystemChannels.platform.invokeMethod('SystemNavigator.
    } else {
      _increment(PopupMenuAction.add1 == item
          ? 1
          : PopupMenuAction.add10 == item ? 10 : 100);
    }
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(
          title: new Text(widget.title),
          actions: <Widget>[
            PopupMenuButton<PopupMenuAction>(
```

```
      onSelected: _onPopupMenuSelected,
      itemBuilder: (BuildContext context) =>
          <PopupMenuEntry<PopupMenuAction>>[
            const PopupMenuItem<PopupMenuAction>(
              value: PopupMenuAction.add1,
              child: Text('+1'),
            ),
            const PopupMenuItem<PopupMenuAction>(
              value: PopupMenuAction.add10,
              child: Text('+10'),
            ),
            const PopupMenuItem<PopupMenuAction>(
              value: PopupMenuAction.add100,
              child: Text('+100'),
            ),
            const PopupMenuDivider(),
            const PopupMenuItem<PopupMenuAction>(
              value: PopupMenuAction.exit,
              child: Text('Exit'),
            ),
          ],
    )
  ],
),
body: new Center(
  child: new Column(
```

```
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            new Text(
              'You have pushed the button this many times:',
            ),
            new Text(
              '$_counter',
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ));
  }
}
```

Radio

This is a material design button that allows the user to select one item from a group of items.
We will cover this in detail in the Forms chapter.

SnackBar

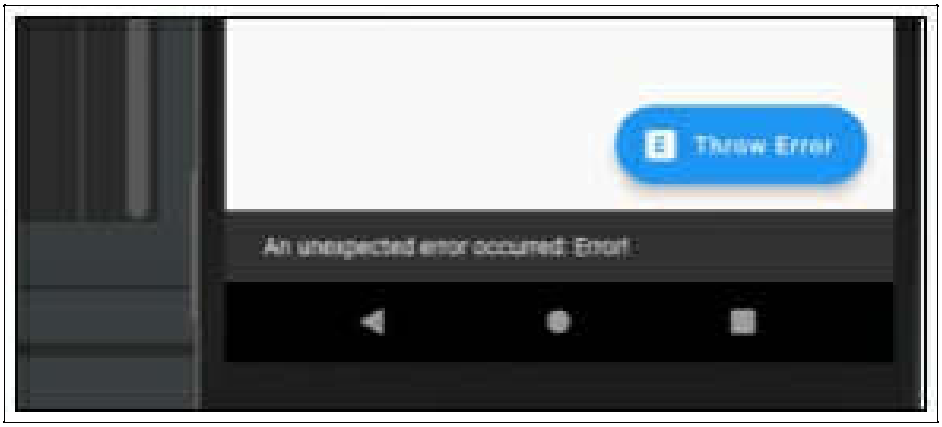
Very useful for showing quick messages to the user, things like:

- Customer deleted.
- Error messages.

Snackbars close themselves, so they don’t leave any unnecessary clutter in the UI.

Example – ‘snack_bar’

This app has a button to simulate an error being displayed with a Snack Bar.



Source Code

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomePageWidget(),
    );
  }
}

class HomePageWidget extends StatelessWidget {
```

```
  HomePageWidget({Key key}) : super(key: key);
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();

  _showSnackBar() {
    _scaffoldKey.currentState.showSnackBar(SnackBar(
      content: Text('An unexpected error occurred: Error!'),
    ));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        key: _scaffoldKey,
        appBar: new AppBar(
          title: new Text("Snackbar"),
        ),
        body: new Center(
          child: new Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              new Text(
                'Content goes here.',
              ),
            ],
          ),
        ),
```

```
      floatingActionButton: new
FloatingActionButton.extended(
          icon: Icon(Icons.explicit),
          label: Text("Throw Error"),
          onPressed: () => _showSnackBar(),
          tooltip: 'Throw Error'));
  }
}
```

Spacer

Spacers can be used to tune the spacing between widgets in a Flex container, like Row or Column. Spacers can be used vertically or horizontally. Spacers sometimes behavior differently from expected as they are not of a fixed width. They attempt to use up all the available space, using the flex property in a similar manner to other Widgets.

Flex Property

The ‘flex’ property (and constructor argument) lets you specify their relative size. Example: a Spacer(flex:5) will be 5 times wider than a Spacer(flex:1).

Exercise – ‘spacer’

We create a basic app with the toolbar icons spaced out using the Spacer Widget.

- You can specify the ‘title’ as a toolbar property. However, if you specify the ‘actions’ property in the toolbar then the title passed in by the ‘title’ property becomes invisible. So, we add the title text to the list of widgets in ‘actions’ property.
- We use several Spacers in the list of widgets in ‘actions’ property. Note that all the spacers are the same size, except the one after the title text, which has a flex of 5. This value tells it to make it 5 times as wide as the others.

Source Code

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomePageWidget(),
    );
  }
}

class HomePageWidget extends StatelessWidget {
  HomePageWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
        appBar: new AppBar(actions: <Widget>[
          Spacer(),
          Center(
              child: Text(
```

```
          "Spacer",
          style: TextStyle(fontSize: 20.0),
        )),
        Spacer(flex: 5),
        IconButton(icon: Icon(Icons.settings_overscan), onPressed: () => {}),
        Spacer(),
        IconButton(icon: Icon(Icons.settings_overscan), onPressed: () => {})
      ]),
      body: new Center(
        child: new Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            new Text(
              'Dummy',
            )
          ],
        ),
      ));
  }
}
```

Switch

This is a material design widget that allows the user to select a yes / no.

We will cover this in detail in the Forms chapter.

TabBar, Tabs and TabBarView Widgets

These Widgets are great for quickly building tabbed user-interfaces. Flutter gives all the Widgets we need to get started with a tabbed interface in just a couple of minutes.

However, you need to bear in mind that these tabs will probably not match you are expecting to see on iOS devices, as these Widgets follow the Google Material design language:

TabBar & TabBarView vs iOS Tabs

	TabBar & TabBarView	**iOS Tabs**
		
Selected item text color	None	Blue
Selected line	Blue	None

(indicator)		
Padding	More	Less

Instructions:

1. Add a TabController. The Controller serves to link the TabBar and TabBarView together. When creating a TabBar, you must either provide a TabController using the "controller" property, or you must ensure that there is a DefaultTabController above the TabBar in the Widget hierarchy.
2. Add a TabBar at the top or the bottom of the Widget. This is the Widget that displays its child Widgets (Tabs) in a bar for selection purposes.
3. Add a TabBarView to the main area of the Widget.

Exercise – ‘tabs_simple’

Let’s build the simplest possible app with a simple tabbed interface containing 3 cat pictures. Then we will modify the tab bar.

Step 1 – Create Default Flutter App

Follow the instructions in Generate Your First App
Leave project open.

Step 2 – Replace Application Code

Replace contents of file ‘main.dart’ in folder ‘lib’ with the following:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class Tab1 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return
Image.network("https://cdn2.thecatapi.com/images/MTY1NDA3
  }
}

class Tab2 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return
Image.network("https://cdn2.thecatapi.com/images/68j.jpg");
```

```
  }
}

class Tab3 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return
Image.network("https://cdn2.thecatapi.com/images/ece.jpg");
  }
}

class HomeWidget extends StatelessWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
        length: 3,
        child: new Scaffold(
            appBar: new AppBar(
              title: new Text("Cat Tabs"),
              bottom: TabBar(
                tabs: <Widget>[
                  Tab(text: 'Cat #1', icon:
Icon(Icons.keyboard_arrow_left)),
                  Tab(text: 'Cat #2', icon:
Icon(Icons.keyboard_arrow_up)),
                  Tab(text: 'Cat #3', icon:
```

```
Icon(Icons.keyboard_arrow_right))
              ],
            ),
          ),
          body: TabBarView(
            children: <Widget>[Tab1(), Tab2(), Tab3()],
          )));
  }
}
```

Step 3 – Open Emulator & Run

Follow the instructions in Open Android Emulator & Run Your First App

Your tabbed interface should appear at the top and look like this:

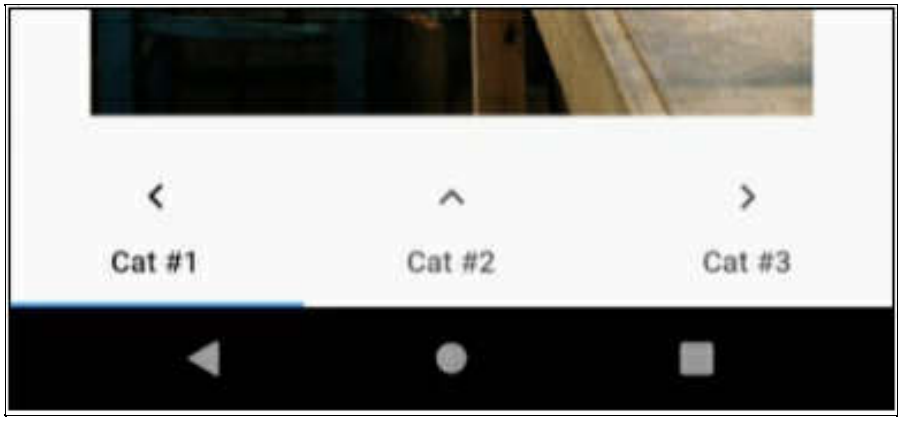
Step 4 – Move Tabs to Bottom

Now let's amend the 'build' code to show the tabs at the bottom to make it look a bit more like the iOS tabs. Change the 'build' method to the following:

```
  @override
```

```
  Widget build(BuildContext context) {
    return DefaultTabController(
        length: 3,
        child: new Scaffold(
            appBar: new AppBar(
              title: new Text("Cat Tabs"),
            ),
            body: TabBarView(
              children: <Widget>[Tab1(), Tab2(), Tab3()],
            ),
            bottomNavigationBar: Container(
                child: TabBar(labelColor: Colors.black, tabs:
<Widget>[
              Tab(text: 'Cat #1', icon:
Icon(Icons.keyboard_arrow_left)),
              Tab(text: 'Cat #2', icon:
Icon(Icons.keyboard_arrow_up)),
              Tab(text: 'Cat #3', icon:
Icon(Icons.keyboard_arrow_right))
            ])));
  }
```

Hot-reload your app and your tabbed interface should appear at the bottom and look like this:



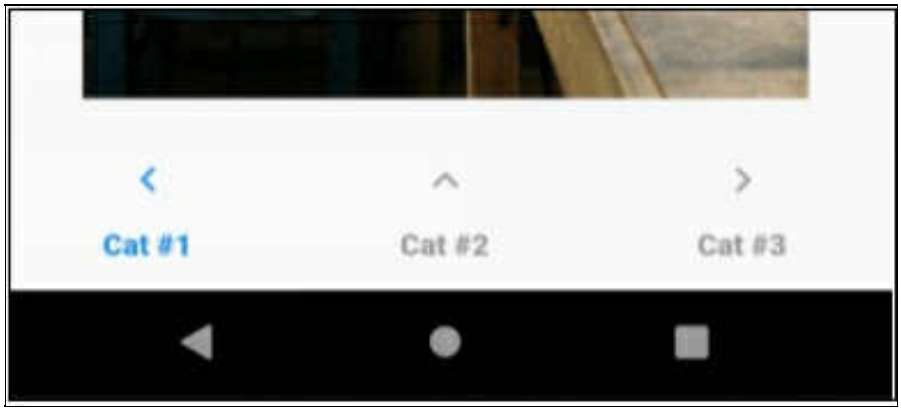
Step 5 – Change Tab Styles To Look More Like iOS

Now let's amend the 'build' code to make the tabs at the bottom look even more similar to those on iOS tabs, without using the Cupertino Widgets.
Change the 'build' method to the following:

```
@override
Widget build(BuildContext context) {
 return DefaultTabController(
    length: 3,
    child: new Scaffold(
      appBar: new AppBar(
        title: new Text("Cat Tabs"),
      ),
      body: TabBarView(
        children: <Widget>[Tab1(), Tab2(), Tab3()],
      ),
```

```
          bottomNavigationBar: Container(
              child: TabBar(
                  labelColor: Colors.blue,
                  unselectedLabelColor: Colors.grey,
                  labelStyle: TextStyle(
                      color: Colors.blue, fontWeight:
FontWeight.w800),
                  indicatorColor: Colors.white,
                  tabs: <Widget>[
                Tab(text: 'Cat #1', icon:
Icon(Icons.keyboard_arrow_left)),
                Tab(text: 'Cat #2', icon:
Icon(Icons.keyboard_arrow_up)),
                Tab(text: 'Cat #3', icon:
Icon(Icons.keyboard_arrow_right))
              ])));
  }
```

Hot-reload your app and your tabbed interface at the bottom should now look like this:



Table

Introduction

The Table Widget works well when you have dynamically-sized components, generated from user data, some wide, some narrow. This widget gives you a great deal of control over column widths (see below).

Column Width Specifiers

The Table Widget has a ‘columnWidths’ argument available in the constructor, which you can populate with a map of column indexes and TableColumnWidth objects. Table Column Width objects can be any of the following:

- FixedColumnWidth
- FlexColumnWidth (attempts to take up a share of the spare width)
- FractionColumnWidth (takes a fraction of the width)
- IntrinsicColumnWidth (sizes the column according to the intrinsic dimensions of all the cells in that column).
- MaxColumnWidth
- MinColumnWidth

Example – ‘table’

This example shows a table with differently sized columns, text wrapping, as well as scrolling.

Source Code

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

void main() => runApp(new TableApp());

class TableApp extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    return new MaterialApp(
      title: 'Flutter Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new HomeWidget(),
    );
  }
}

class HomeWidget extends StatelessWidget {
  HomeWidget({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    const TableRow tableRow = TableRow(children: [
      const Text("aaaaaaaaaaaaaaaaaaaa", overflow:
TextOverflow.fade),
      const Text("bbbbbbbbbbbbbbbbbbbb", overflow:
TextOverflow.fade),
      const Text("ccccccccccccccccccccc", overflow:
TextOverflow.ellipsis)
    ]);
    return new Scaffold(
        appBar: new AppBar(title: new Text("Table")),
        body: new Table(
```

```
        children: [
          tableRow,
          tableRow,
          tableRow,
          tableRow,
          tableRow,
          tableRow,
          tableRow,
          tableRow,
          tableRow,
        ],
        columnWidths: const <int, TableColumnWidth>{
          0: FlexColumnWidth(0.1),
          1: FlexColumnWidth(0.3),
          2: FlexColumnWidth(0.6),
        },
        border: TableBorder.all(),
      ) // end table,
      );
  }
}
```