<https://flutter.dev/docs/development/ui/widgets-intro>

# Widgets

The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget’s state changes, the widget rebuilds its description.

flutter create --androidx -t app --org com.ahcompany.guide -a java -i swift welcome

# Welcome App

The minimal Flutter app simply calls the runApp() function with a widget:

import 'package:flutter/material.dart';

void main() {

runApp(

Center(

child: Text(

'مرحبا',

textDirection: TextDirection.rtl,

style: TextStyle(

fontSize: 40.0,

fontWeight: FontWeight.bold,

),

),

),

);

}



The runApp() function takes the given Widget and makes it the root of the widget tree. In this example, the widget tree consists of two widgets, the Center widget and its child, the Text widget.

When writing an app, you’ll commonly author new widgets that are subclasses of either StatelessWidget or StatefulWidget, depending on whether your widget manages any state. A widget’s main job is to implement a build() function, which describes the widget in terms of other, lower-level widgets.

# Basic widgets

Flutter comes with a suite of powerful basic widgets, of which the following are commonly used:

Text

The Text widget lets you create a run of styled text within your application.

Row, Column

These flex widgets let you create flexible layouts in both the horizontal (Row) and vertical (Column) directions. The design of these objects is based on the web’s flexbox layout model.

Stack

Instead of being linearly oriented (either horizontally or vertically), a Stack widget lets you place widgets on top of each other in paint order. You can then use the Positioned widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web’s absolute positioning layout model.

Container

The Container widget lets you create a rectangular visual element. A container can be decorated with a BoxDecoration, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size. In addition, a Container can be transformed in three dimensional space using a matrix.

Below are some simple widgets that combine these and other widgets:

import 'package:flutter/material.dart';

class MyAppBar extends StatelessWidget {

MyAppBar({this.title});

// Fields in a Widget subclass are always marked "final".

final Widget title;

@override

Widget build(BuildContext context) {

return Container(

height: 120.0, // in logical pixels

padding: const EdgeInsets.only(top:50.0),

decoration: BoxDecoration(color: Colors.blue[500]),

// Row is a horizontal, linear layout.

child: Row(

// <Widget> is the type of items in the list.

children: <Widget>[

IconButton(

icon: Icon(Icons.menu),

tooltip: 'Navigation menu',

onPressed: null, // null disables the button

),

// Expanded expands its child to fill the available space.

Expanded(

child: title,

),

IconButton(

icon: Icon(Icons.search),

tooltip: 'Search',

onPressed: null,

),

],

),

);

}

}

class MyScaffold extends StatelessWidget {

@override

Widget build(BuildContext context) {

// Material is a conceptual piece of paper on which the UI appears.

return Material(

// Column is a vertical, linear layout.

child: Column(

children: <Widget>[

MyAppBar(

title: Text(

'Example title',

style: Theme.of(context).primaryTextTheme.headline6,

),

),

Expanded(

child: Center(

child: Text('Welcome'),

),

),

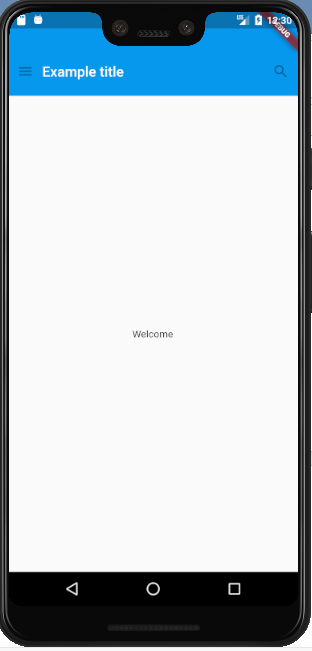
],

),

);

}

}



The MyAppBar widget creates a Container with a height of 120 device-independent pixels with an internal padding of 50 pixels, from the top. Inside the container, MyAppBar uses a Row layout to organize its children. The middle child, the title widget, is marked as Expanded, which means it expands to fill any remaining available space that hasn’t been consumed by the other children. You can have multiple Expanded children and determine the ratio in which they consume the available space using the flex argument to Expanded.

The MyScaffold widget organizes its children in a vertical column. At the top of the column it places an instance of MyAppBar, passing the app bar a Text widget to use as its title. Passing widgets as arguments to other widgets is a powerful technique that lets you create generic widgets that can be reused in a wide variety of ways. Finally, MyScaffold uses an Expanded to fill the remaining space with its body, which consists of a centered message.

# Using Material Components

Flutter provides a number of widgets that help you build apps that follow Material Design. A Material app starts with the MaterialApp widget, which builds a number of useful widgets at the root of your app, including a Navigator, which manages a stack of widgets identified by strings, also known as “routes”. The Navigator lets you transition smoothly between screens of your application. Using the MaterialApp widget is entirely optional but a good practice.

import 'package:flutter/material.dart';

void main() {

runApp(MaterialApp(

title: 'Flutter Tutorial',

home: TutorialHome(),

));

}

class TutorialHome extends StatelessWidget {

@override

Widget build(BuildContext context) {

// Scaffold is a layout for the major Material Components.

return Scaffold(

appBar: AppBar(

leading: IconButton(

icon: Icon(Icons.menu),

tooltip: 'Navigation menu',

onPressed: null,

),

title: Text('Example title'),

actions: <Widget>[

IconButton(

icon: Icon(Icons.search),

tooltip: 'Search',

onPressed: null,

),

],

),

// body is the majority of the screen.

body: Center(

child: Text('Welcome'),

),

floatingActionButton: FloatingActionButton(

tooltip: 'Add', // used by assistive technologies

child: Icon(Icons.add),

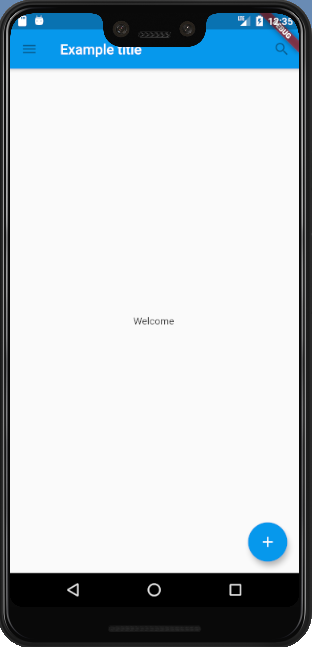
onPressed: null,

),

);

}

}



Now that the code has switched from MyAppBar and MyScaffold to the AppBar and Scaffold widgets, and from material.dart, the app is starting to look at bit more Material. For example, the app bar has a shadow and the title text inherits the correct styling automatically. A floating action button is also added.

Notice that widgets are passed as arguments to other widgets. The Scaffold widget takes a number of different widgets as named arguments, each of which are placed in the Scaffold layout in the appropriate place. Similarly, the AppBar widget lets you pass in widgets for the leading widget, and the actions of the title widget. This pattern recurs throughout the framework and is something you might consider when designing your own widgets.

# Handling gestures

Most applications include some form of user interaction with the system. The first step in building an interactive application is to detect input gestures. See how that works by creating a simple button:

# How do I set the background color of my main screen in Flutter?

There are many ways of doing it, I am listing few here.

1. Using backgroundColor
2. Scaffold(
3. backgroundColor: Colors.black,
4. body: Center(...),

)

1. Using Container in SizedBox.expand
2. Scaffold(
3. body: SizedBox.expand(
4. child: Container(
5. color: Colors.black,
6. child: Center(...)
7. ),
8. ),

)

1. Using Theme
2. Theme(
3. data: Theme.of(context).copyWith(scaffoldBackgroundColor: Colors.black),
4. child: Scaffold(
5. body: Center(...),
6. ),

)