Flutter SDK

A guide for getting started in Flutter

Writers: Henrik Rosander and Linn Storesund

Introduction

Flutter is a Software Development Kit (SDK) for the programming language Dart. This guide will help you get started with how to configure Flutter for your own computer and go through the basics of Flutter.

Installing Flutter

To install Flutter, go to the website https://www.flutter.dev/, and download the SDK matching your operating system. Unzip the downloaded file and save it at a reasonable location (Note: Do not save it where privileged authorization is needed, such as C:/Users in Windows or Root on Linux.)

Devices

Flutter can be run through both emulators, either iPhone or Android, but also as a web application. One free emulator with an IDE for Flutter is Android Studio, which can be downloaded here: https://developer.android.com/studio/.

Using Flutter's beta version, Flutter can be run through the browser; which can be found here: https://flutter.dev/docs/get-started/web/.

When the platform that you are going to develop on has been decided, follow the steps to run Flutter.

Configuring Flutter to run in Android Studio

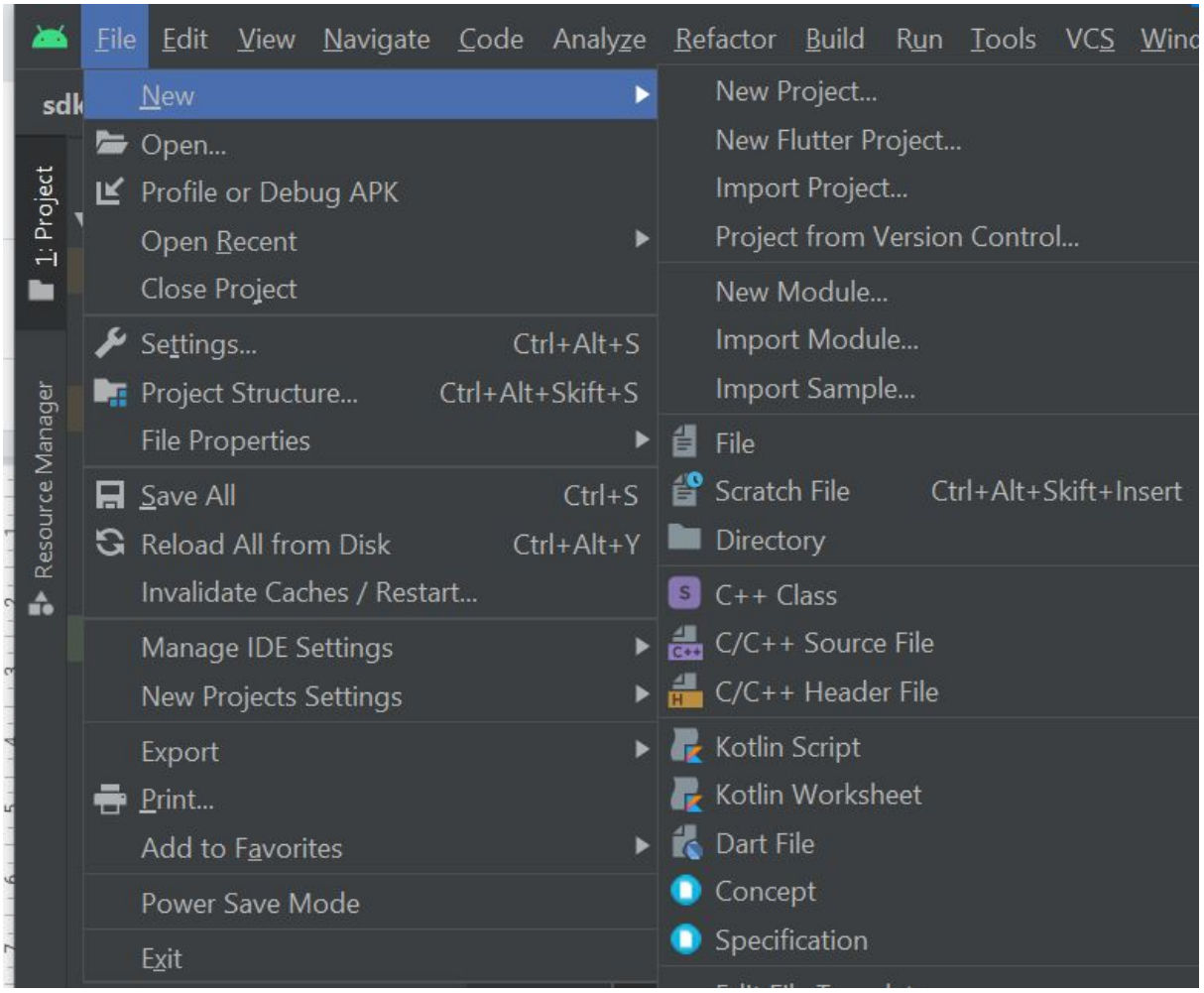
Go to File->Settings->Plugins and search Flutter to install the integrated Flutter support in Android Studio.

Run flutter through the command prompt (Windows)

The most simple way to run Flutter through the command prompt is to add the location of your flutter bin folder to the environmental variable paths in your OS. Search for environmental variables in the windows search bar and navigate to path to update it with the location of your Flutter SDK bin (.../Flutter/bin). After this, you should be able to directly use flutter in the command prompt by writing "flutter -Command-".

Create your own Flutter application

To create a new app you should now after installing flutter, have the opportunity to click on (File -> New -> New Flutter Project) on Android Studio application, see figure 1. If you want to do an application from scratch, you pick “Flutter Application” then all the required files for the application will be created. The file you will modify to get as you want is the file “main.dart” which you find in the folder lib. If you want to upgrade systems or add images, fonts etc., you will change it in the file “pubspec.yaml”.



The first default file has a lot of code that creates a counter that you can compile if you choose a device (a phone or a web browser) and then press run. The phone can take some time so be patient for the first time! You can now remove or modify some of the default code but first it's good to know what different parts are used for.

Widgets

The concept of Flutter is that almost everything is a widget, even layout models are widgets as text, buttons and icons but also widgets that we don't see such as rows and columns. But first look at your default code in *main.dart* which contains a Stateless widget and a Stateful widget and these are good to know about.

The Stateless widget cannot change during the application's running and can be used if you want a title or a theme for your application. You can in the default code change the title to what you want and then perhaps change the theme color. You can choose between a lot of colors if you write "Colors." and a drop down menu with colors will appear. Also the color you pick will be shown in the left column. Here is an example for a Stateless Widget:

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

The Stateful widget is the homepage of your application. It is stateful, means the class is the configuration for the state and can be updated and changed during the application is running.

The Layout Widgets

When you design your application interface you create a layout by composing widgets to build more complex widgets. All widgets have different properties for example height if it's a layout widget or text color if it's a text widget.

The first thing to do is to choose a layout widget based on how you want to align your content, for example you can choose '*Center'* which centers your application. After that you can add both visible widgets as button widget or an invisible widget as *´Row´* and *´Column´* for horizontal and vertical layouts. All widgets have either a child property if it is going to contain only one widget or children property if they take a list of widgets. For an example if you want to have text and a button next to each other you implement these as children in a

´Row´ widget or if you want an image in the middle of your application you implement the widget as a child in a *´Center'* widget.

Callback Functions

Use controllers and in the classes that use states the controller can have listeners added, meaning they are checking when the widgets that use it.

Another way of having the application reacting to widgets is to use the setState function, which can be used directly for widgets like buttons, and through controllers.

By using a stateful widget, values can be updated within the application. As the figure below demonstrates, where we use a 'onPressed'-callback, the program will call upon the code in myFunction() when the RaisedButton is being pressed.

```
class stateExample extends StatefulWidget {
  @override
  _stateExampleState createState() => _stateExampleState();
}

class _stateExampleState extends State<stateExample> {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        RaisedButton(
          onPressed: myFunction(),
        ) // RaisedButton
      ], // <Widget>[]
    ); // Column
  }

  myFunction() {
    //Your code goes here...!
  }
}
```

Another example is to use controllers, which always gets the value from the property of the widget. This can be used to update variables like shown in the picture below. The picture below illustrates an example with a TextFormField, a text field for the user to enter text into.

This code uses setState(), which notifies the framework that the state of this property is or has been changed and updates the variable textInput accordingly.

```
String textInput;

class _TextReaderState extends State<TextReader> {
  final textController = TextEditingController();

  void initState() {
    super.initState();
    textController.addListener(() {
      setState(() {
        textInput = textController.text;
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    return TextFormField(
      controller: textController,
    );
  }
}
```

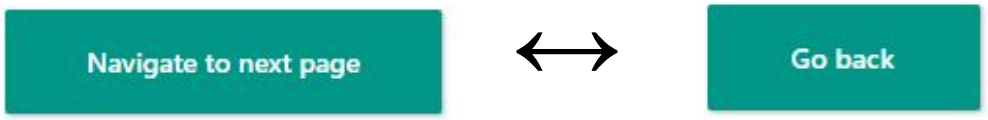
Navigation

Using the navigator prop we can navigate between different pages on our application. This is done by managing the stack, which the navigator does. To navigate, we can either use Navigator.pages, or, as seen in the figure below, use navigator.push and navigator.pop. This works by pushing one page on to the stack, then popping it once we need to go back.

```dart
class _FirstState extends State<FirstPage> {
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        RaisedButton(
            color: Colors.teal,
            padding: EdgeInsets.only(left: 50, right: 50, top: 25, bottom: 25),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => NextPage()),
              );
            },
            child: Text("Navigate to next page",
                style: TextStyle(color: Colors.white))),  // Text, RaisedButton
      ],  // <Widget>[]
    );  // Column
  }
}

class _NextPageState extends State<NextPage> {
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Column(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[
            RaisedButton(
              color: Colors.teal,
              padding:
                  EdgeInsets.only(left: 50, right: 50, top: 25, bottom: 25),
              onPressed: () {
                Navigator.pop(context);
              },
              child: Text("Go back", style: TextStyle(color: Colors.white)),
            ),  // RaisedButton
          ],  // <Widget>[]
        )  // Column
      ],  // <Widget>[]
    );  // Column
  }
}
```

Which leads to this result when compiled:



We are hoping this introduction guide helps you to get to know Flutter and now you can create amazing Flutter applications!