# MY FLUTTER SUMMARY

***For now this text will be completely helpful to windows users.***

**INTRODUCTION:**

Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. Flutter works with existing code. It is used by developers and organizations around the world, and is free and open source.

Dart is the programming language flutter uses, so you are advised to learn the programming language before the framework( Flutter). It will help you to learn faster with much confidence. You can us my dart Video at **url** for better learning.

**TERMINOLOGIES**

**PascalCase** means UpperCamelCase.

**INSTALLATION**

**INSTALLING FLUTTER SDK:**

1. To install and run Flutter, your **development environment** must meet these **minimum** requirements:

* **Operating Systems**: Windows 10 or newer (64-bit), x86-64 based.
* **Disk Space**: 1.64 GB ( Which your IDE or Other tools are not part of) but you shouldn’t bother about this because most recent PC have enough memory.
* **Tools**: Flutter depends on these tools being available in your environment.
  + [Windows PowerShell 5.0](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell) or newer (this is pre-installed with Windows 10)
  + [**Git for Windows**](https://git-scm.com/download/win) 2.x, with the **Use of Git from the Windows Command Prompt** option.

*If Git for Windows is already installed, make sure you can run****git****commands from the* ***command prompt*** *and* ***PowerShell****.*

1. Click on the link -- [**https://storage.googleapis.com/flutter\_infra\_release/releases/stable/windows/flutter\_windows\_3.0.2-stable.zip**](https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_3.0.2-stable.zip) -- to download the flutter sdk in your windows PC.
2. After downloading, go to your **C drive** and create a new folder called anything you like but I will call mine **src**;

Example **c:/src**

***Caveat:***

*Do not install Flutter to a path with name that contains* ***special characters or spaces****. Also, Do not install Flutter in a directory* ***like C:\Program Files\****that requires elevated privileges*

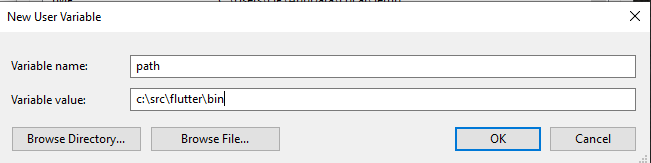
1. Now go to **your PC downloads( where the flutter zip file was downloaded to)** and extract the **flutter zip** file to the **c:/src** you just created now.
2. After the extraction, go to **bin folder** inside the **flutter sdk** folder you just extracted now, copy the path; example: **c:\src\flutter\bin**.
3. Go to your **pc search** and type **envir**, then click to **edit your environmental variable**
4. Clicking **Environmental variable**…

* under **users variable**, edit your **path** variable if you already have a **path** variable created before. This can be done by  appending the the copied location of **flutter\bin** in **no.5** above using **“;”** as a separator from existing values.

or else

* click on **new**
* then type **path** as **variable name**
* and paste the **path** you copied at **no.5** above as **variable value**.
* then press **enter/return** key on the keyboard to **save** the changes.

Example:



***Caveat****:*

*Flutter* ***1.19.0 dev release*** *and* ***above******adds******dart SDK*** *alongside the* ***flutter SDK****. If you already downloaded* ***the dart SDK separately****, make the* ***path to your flutter SDK*** *set in your* ***users variable*** *above to be* ***before*** *the* ***dart SDK****.*

*To check if* ***flutter SDK comes first*** *type in your terminal* ***where flutter dart(*** *Or* ***where.exe flutter dart*** *if you are using powershell****)****. You will see output like below:*

*C:\>* ***where flutter dart***

*C:\src\flutter\bin\flutter*

*C:\src\flutter\bin\flutter.bat*

*C:\src\flutter\bin\dart*

*C:\src\flutter\bin\dart.bat*

*C:\src\dart-sdk\bin\dart.exe*

1. After you must have saved and closed the new changes you made in your **environmental variable**, Open your command line and type:

* **Flutter** then hit **enter/return** button to see some code to shows that you have successfully installed **Flutter**. This will also show you some commands that are useful.
* **Flutter --version** then hit enter button to see some code to shows the version of flutter you installed.
* **flutter doctor** to know the IDEs and tools you need to install before creating your flutter App.
* **flutter doctor -v** to see more information on what you need to install and those u have installed.
* **flutter create app\_name** to create a flutter most basic App.
* **flutter upgrade** to upgrade your flutter SDK.
* **flutter config –no-analytics** to disable flutter reporting some of its usage details( like usage, crashes etc.) to google.
* **flutter config** displays the current setting of the above.
* **flutter config –andriod-studio-dir <directory>** is used to set the directoy that Andriod studio is installed to.

1. To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) or higher.

**FAULTS:**

1. **‘flutter’ is not recognized as an internal…**

* Set your path **…\flutter\bin**.

1. **‘where’ is not recognized as an internal…**

* Go to **C:\Windows\System32** and copy it’s path, paste it in your users variable of your **environment variables**. You can still copy the above, it is the same thing.

1. **Error: unable to find git in your path…**

* Set your **…\git.exe** path and **…\git\cmd** path.

1. **PowerShell.exe is not recognized as an internal…**

* Set your **…\PowerShell-7.1.2-win-x64** path.

1. DONE!.

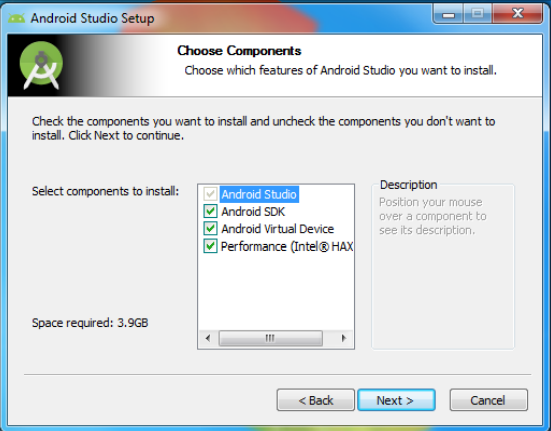
**INSTALLING ANDRIOD STUDIO:**

1. Android studio is essential because it comes with some functionalities needed to create Android Apps.
2. [**https://developer.android.com/studio**](https://developer.android.com/studio) click this link to download Android studio.
3. Accept the terms and conditions that google gave before you can proceed with the downloading.
4. Open the folder where you have downloaded Android Studio. **Right-click** on the **Android Studio exe** file and select Run as administrator.
5. When you are being ask to allow **Android Studio** to make changes in your **PC**, click **yes** to start the set-up wizard.
6. Now, let’s run through the android studio set-up wizard windows step-by-step. Watch each of my set-up windows carefully to select those things I selected in mine before clicking next. Don’t bother if there is any set-up window that you did see because Andriod Studio have different versions. Better still read the write-up in each window to know what is necessary for you.

* Step1:

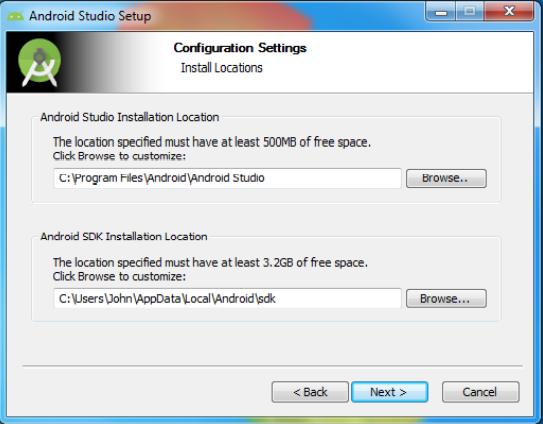


* Step2:

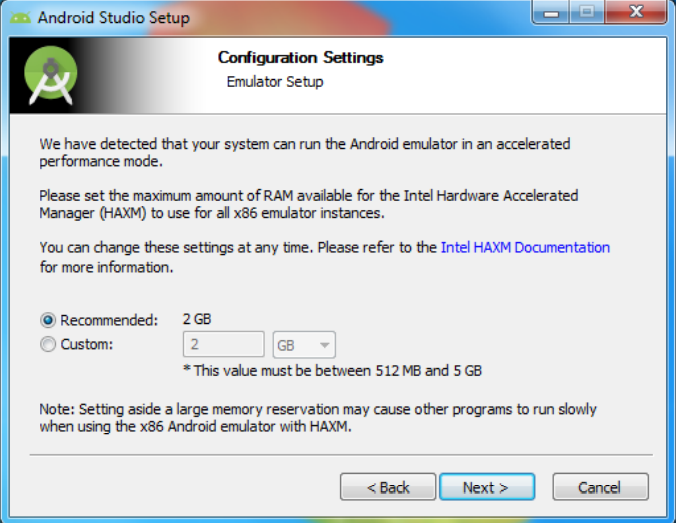


* Step3:

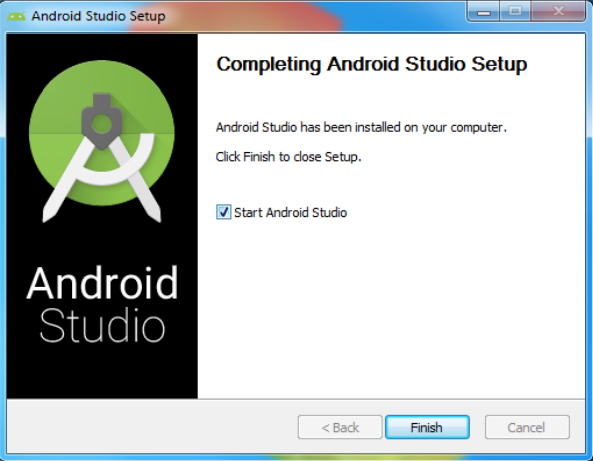
Choose these locations below wisely, mine will be where my flutter and Dart SDK is located; that’s **C:/src** file.



* Step4:

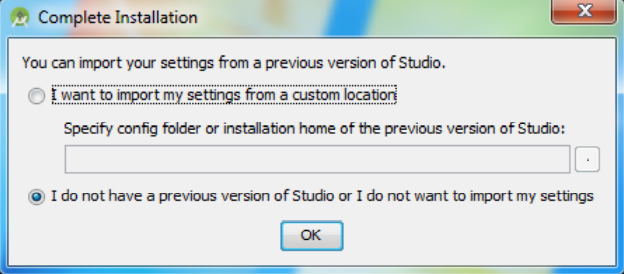


* Step5:



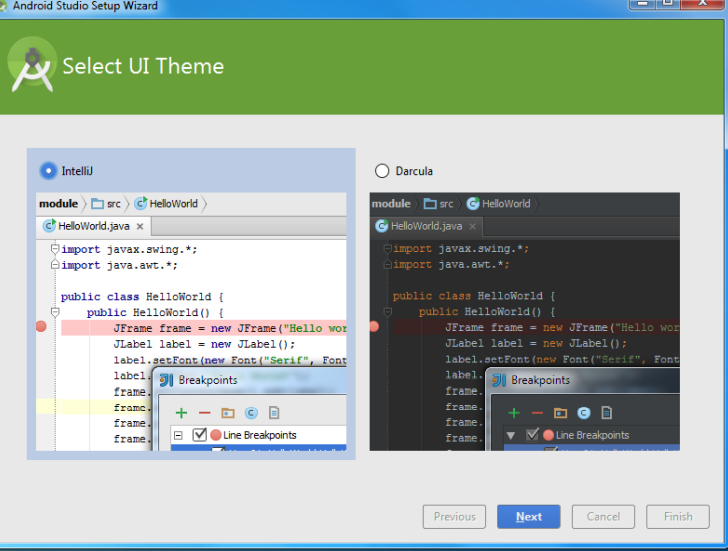
* Step6:

If this is your first time using Android Studio, click on the I do not have a **previous version of Android Studio** or **I do not want to import my settings radio button** and click on **OK**.



* Step7:

In the Below set-up window, If you like a conventional black text on white background appearance, then choose **IntelliJ**, and if you want a cool dark style, choose **Darcula**.



* After the set-ups, Android Studio will connect to the internet to download some tools( like Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools) that we need. Again, this could take a while depending on how fast your internet connection is because the files are large.
* When the Downloading is through, click on **finish**.
* Your **Android SDK** is now ready,so you can create new project or open an exiting project from the android studio.

1. Run **flutter doctor** to confirm that Flutter has located your installation of Android Studio. If Flutter cannot locate it, run **flutter config --android-studio-dir <directory>** to set the directory that you installed the Android Studio.

**Example**: **flutter config --android-studio-dir C:\src\Andriod** because I installed my android studio in **C:\src\Android**.

1. Done!

Caveat:

*Make sure that you have a version of* ***Java 8 SDK*** *or higher installed and that your****JAVA\_HOME******environment variable*** *is set to the JDK’s folder( Android Studio versions 2.2 and above come with this JDK, so this should already be done).*

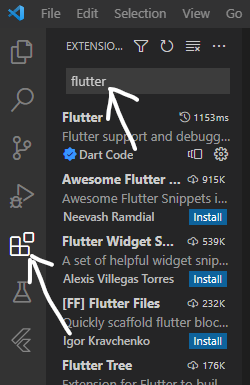
*Run* ***flutter doctor --android-licenses*** *to start signing licenses.*

*Review the terms of each license carefully before agreeing to them.*

**CREATING, RUNNING, TESTING, COMPILING AND DEBUGGING A NEW FLUTTER APP:**

**CREATING A FLUTTER APP:**

To create a flutter App, you need to install flutter and dart plugins. The image below is a guide on how you can install the flutter plugins. Click on extension-as shown by the lower arrow- , then search flutter in the search bar – as shown by the upper arrow - , then installed the plugins with name flutter, this will install Dart plugin alongside. Or you can install Dart plugin manually to ensure that it is installed.



Now let’s create our first app:

1. You must be **connected to the internet**.
2. Go to your **VScode terminal** or to your **PC command prompt**, and type in **flutter create app\_name**. This process might take some seconds before it will complete the App creation depending on your internet connection strength.
3. Dart packages and flutter Apps names must be in small letters, which uses underscore to separate 2 or more words that is contained in the name.
4. **DONE!**.

**RUNNING A FLUTTER APP:**

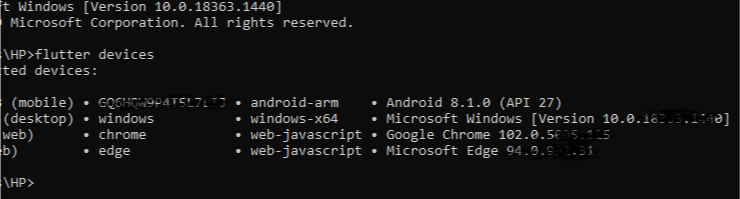
You can run flutter Apps on **web, windows, emulator, real mobile device, mac OS, Linux, chrome OS** etc. All these platform have some requirement needed to run flutter Apps. **Web flutter App** is the easiest, it just need a browser, so I will not discuss it in details.

**DEPLOYING YOUR APP TO A REAL DEVICE:**

* To Run flutter for the first time, you need internet connection.
* Enable **Developer options:**

**This can be done by tapping Build number 7 times, Build number is in your about phone, which is in your android device settings.**

* **Now go to Developer Options in your Android device settings and enable USB debugging** .
* **Windows-only**: Install the [**Google USB Driver**](https://developer.android.com/studio/run/win-usb) which I don’t think it is necessary because many recent device just need your device to be plugged in.
* Plug your phone into your computer using USB, then permit your computer to access your android device.
* Run the **flutter devices** command to know if Flutter recognizes your connected Android device. You will see text like in the image below.

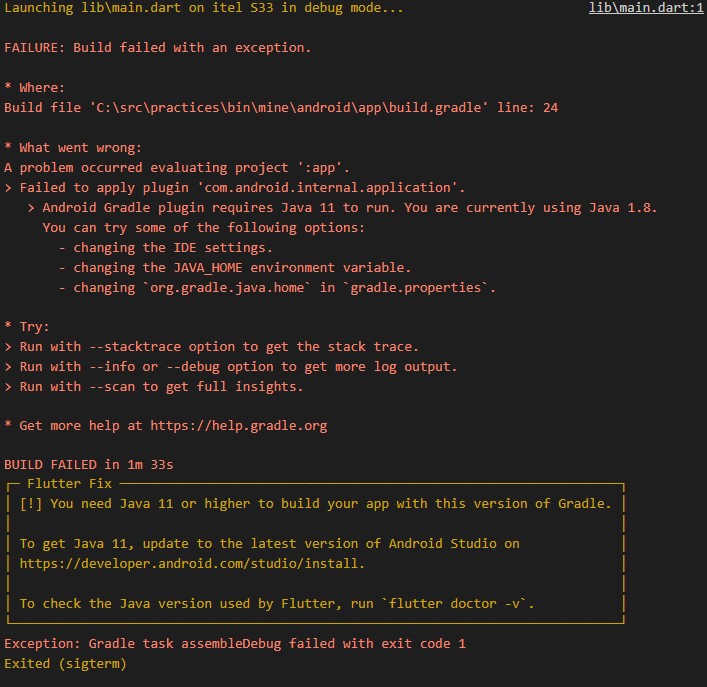


* done!

**Caveat**:

*By default, Flutter uses the version of the Android SDK where your****adb****tool is based but if you want Flutter to use a different installation of the Android SDK, you must set the****ANDROID\_SDK\_ROOT****environment variable to that installation directory.*

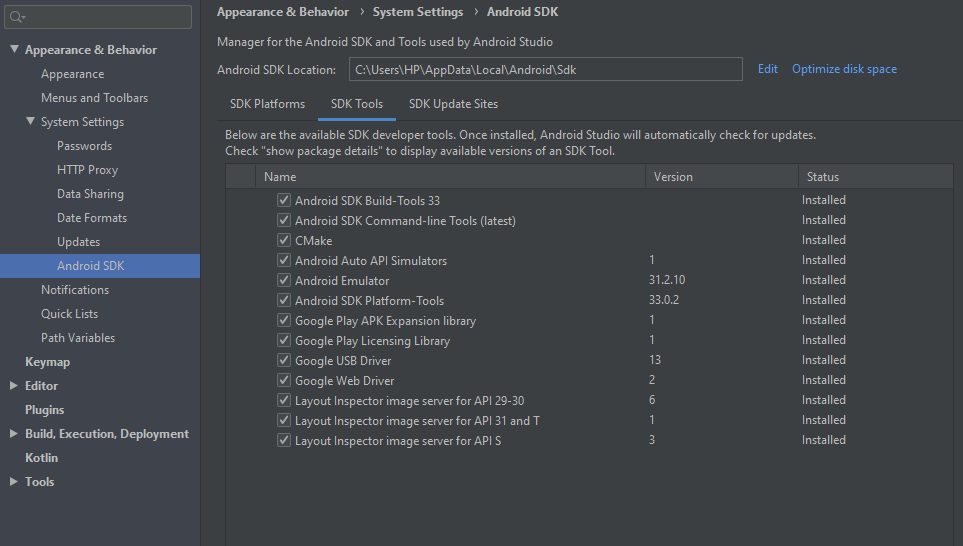
FAULTS:



* Download java jdk that is later than jdk 11
* set your JAVA\_HOME variable with where the above jdk is installed to, **not** the bin in the jdk.
* Also add **%JAVA\_HOME%\bin** to your path variable
* Open your **android studio**, go to **File>project structure** and change the **java jdk** to the one you downloaded.
* Go the where you installed Android studio and delete the jre file and paste the jdk you installed.
* change the jdk name from jdk 11 or jdk 12 or … to jre.
* Now run **flutter doctor –v** to see the jdk version that flutter will run with under your android studio part.
* DONE!

1. **commandline-tools component is missing**

**Run 'path/to/sdkmanager --install "cmdline-tools;latest"'**



Complete your installation in android studio. Even if it works earlier.

1. DONE!.

**STATE MANAGEMENT**

Stateless widgets are immutable, meaning that their properties can’t change—all values are final.

Stateful widgets maintain **state** that **might change** during the lifetime of the widget. Implementing a stateful widget requires at least two classes: 1) a StatefulWidget class that creates an instance of 2) a State class. The StatefulWidget class is, itself, immutable and can be thrown away and regenerated, but the State class persists over the lifetime of the widget.

**WIDGETS**

1. **CIRCULARPROGRESSINDICATOR**:

**CircularProgressIndicator** is used to show the progress of a file loading( mostly from internet).

The value parameter when set to null makes the loading progress **indeterminate** but when set to a value between **0.0** and **1.0**, it will load depending on the download progress or the delay progress you set it to.

1. **LINEARPROGRESSINDICATOR**:

**LinearProgressIndicator** is used to show the progress of a file loading( mostly from internet).

The value parameter when set to null makes the loading progress **indeterminate** but when set to a value between **0.0** and **1.0**, it will load depending on the download progress or the delay progress you set it to.

1. **CLIPOVAL**:

**ClipOval** is a widget used to place your image in a circle or oval shape depending on the image shape. If the image is a rectangle, it will be clipped in an Oval shape but when it is in a square, it will be clipped in a square shape.

You can also have your own customize clip by extending **CustomClipper<T>** which is an **abtract class**, thus you are required to override two method.   
**getClip(…)** and **shouldReclip(…)**

**Example:**

class Mine extends CustomClipper<Rect> {

  @override

  Rect getClip(Size size) {

    return Rect.fromLTWH(0, 0, 200, 50);

  }

  @override

  bool shouldReclip(covariant CustomClipper<Rect> oldClipper) {

    return true;

  }

}

When the **CustomClipper<Path>** has **Path** as it’s parameter, then you can specify the shape you want your widget to take when overriding **getClip(…)**.

1. **CLIPRRECT:**

**ClipRRect** is used to clip only the border of the widget specified as it’s child. Example:

 ClipRRect(

        borderRadius: BorderRadius.circular(7),

        child: MyWidget(),

      ),

1. **CONSTRAINEDBOX**:

**ConstrainedBox** is used to specify a particular width and height to a widget.

1. **CONTAINER**:

**Container** is a widget that is used to style a widget as you like. It have many parameter that helps you to do this. like:

* **alignment**: //make the contain to be of the same size ass it parents, thus allowing you to place the child widget where you like.
* **padding**:// space between the **Container** and it’s child.
* **margin**://space between the **Container** and parents
* **transform**:// The orientation of the **Container**
* **clipBehavior**: // etc

1. **DATATABLE:**

**DataTable** is used to create a table. It has many parameter which gives more functionalities to the table.like:

* columns://It is a list that contains all the column you want your table to have. Each of this column are added with DataColumn(…) widget like in below example.
* rows:// Specifies the rows that each line has.
* sortAscending:// sorts in ascending other
* sortColumnIndex:// Aids in sorting
* numeric: //Used in DataColumn to …
* showEditIcon:// Used in DataCell to show that a cell is editable.
* //…

Example:

          child: DataTable(

            sortAscending: true,

            sortColumnIndex: 0,

            columns: const [

              DataColumn(

                label: Text("Names"),

                numeric: true,

              ),

              DataColumn(label: Text("Year")),

              //...

            ],

            rows: [

              DataRow(

                cells: [

                  DataCell(

                    Image.asset("url"),

                    showEditIcon: true,

                  ),

                  const DataCell(Text("2018")),

                  //...

                ],

              ),

              const DataRow(

                cells: [

                  DataCell(Text("Gopher")),

                  DataCell(Text("2018")),

                  //...

                ],

              ),

              //...

            ],

          ),

If you know that your table will over flow, add it to a **SingleChildScrollView(…)**

1. **TABBAR AND TABBARVIEW:**

**TabBar** and **TarBarView** are used in flutter to create different tabs and their different pages like in the below code and sample image.

**TabController** is Used to these **TabBar** and **TarBarView** **Maunaully** and **DefaultTabController** is used to add it **automatically**.

void main() {

  runApp(const TabBarDemo());

}

class TabBarDemo extends StatelessWidget {

  const TabBarDemo({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: DefaultTabController(

        length: 3,

        child: Scaffold(

          appBar: AppBar(

            bottom: const TabBar(

              tabs: [

                Tab(icon: Icon(Icons.directions\_car)),

                Tab(icon: Icon(Icons.directions\_transit)),

                Tab(icon: Icon(Icons.directions\_bike)),

              ],

            ),

            title: const Text('Tabs Demo'),

          ),

          body: const TabBarView(

            children: [

              Icon(Icons.directions\_car),

              Icon(Icons.directions\_transit),

              Icon(Icons.directions\_bike),

            ],

          ),

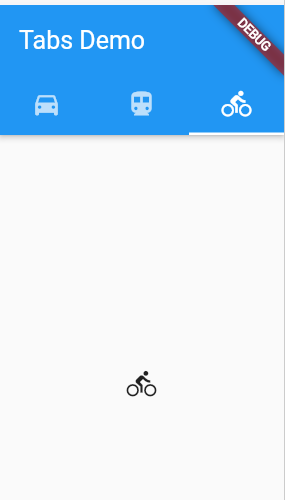
        ),

      ),

    );

  }

}



To add TabBar and TabBarView in Cupertino App(IOS style) visit - [**https://codelabs.developers.google.com/codelabs/flutter-cupertino**](https://codelabs.developers.google.com/codelabs/flutter-cupertino) -.

1. **DISMISIBLE**

**Dismissible** Widget is used in List widgets to sweep the list content of like achieve Chat in Whatsapp.

key and child property are required in Dismissible widget.

1. **DIVIDER**

**Divider** are used to add lines between two widgets especially in Form or List. It has much useful parameters like **height**, **thickness**, … etc.

1. **DRAGGABLE**

**Draggable** widget is used to add Widget that are draggable from one place to another. It have much functionalities like knowing where it left from… etc.

1. **DRAGGABLESCROLLABLESHEET**

**DraggableScrollableSheet** is a widget used to create a page or list that you can drag from bottom to top or vice verse. Look up in flutterdoc.dev.

1. **DRAWER**

**Drawer** Widget is used in **Scaffold** to Add a side bar that be used to select item easily. Read more.

1. **EXPANDED**

Expanded widget is used create a flexible box. it has many useful parameters like flex,… etc.

1. **FADEINIMAGE**

**FadeInImage** is used to save an image before loading in to your app with your choice transition animation.

1. **FADETRANSITION**

**FadeTransition** helps you to used different transition to load in Widgets.

1. **FITTEDBOX**

**FittedBox** Widget is used to make it’s child widget to fit it’s parents Widget as you like.

1. **FLEXIBLE**

**Flexible** Widget is used to create a flexible box. it has useful parameters like flex… etc.

1. **FLOATINGACTIONBUTTON**

**FloatingActionButton** is used to add button like the one you found your default flutter first App, that was used to increment the count.

1. **FRACTIONALLYSIZEDBOX**

**FractionallySizedBox** widget is used to add widgets( mostly button) that you want to be in a particular **percent** with the **App width**.

1. **FUTUREBUILDER**

**FutureBuilder** is used mostly to download file. read up.

1. **HERO**

**Hero** widget is used to create a widget that **when clicked opens a page**, like clicking a bag and it will move you to a page with more info about the bag.

1. **IGNOREPOINTER**

**IgnorePointer** widget is used to make some of your widget not to be clickable.

1. **INDEXSTACK**

**IndexStack** is used to add **automatically scrolling Images**. It is very useful, so read-up.

1. **INTERACTIVEVIEWER**

**InteractiveViewer** is used to add the full detail of a it’s child widget which can be **zoomed** or **contract** by client. it have much useful parameters , so read-up.

1. **LAYOUTBUILDER**

**LayoutBuilder**… Read-up.

1. **LIMITEDBOX**

**LimitedBox** is used to give it’s child a default size. A list can be added to it to constraint the list to a particular size which can be Scrollable.

1. **MEDIAQUERY**

**MediaQuery** is used to get details of the device your app is running on, like the size… etc. These informations can be useful in designing your App.

1. **NOTIFICATIONLISTENER**

**NotificationListener**… read up.

1. **OPACITY**

**Opacity** widget is used mostly to hide widgets with their space not removed.

1. **PADDING**

**Padding** widget is used to pad a child widget in it’s parent widget.

1. **PageLayoutBuilder**
2. **PAGEVIEW**

**PageView** is used to create pages that you want it to be scrolled from one page to another page. It can be to scroll different items and their info, which you place in different pages.

1. **PHYSICALMODEL**

**PhysicalModel** is used to add physical properties to it child widgets like shadow… etc. read-up.

1. **PLACEHOLDER**

**PlaceHolder** is used … read-up.

1. **POSITIONED**

**Positioned** widget is always used as the children of **Stack** widget to place your child widget in any position u want. Positioned widget has many named constructors that are useful.

1. **REORDERABLELISTVIEW**

**ReorderableListView** is used to create a ListView that user can drag and drop it’s children and place each at a more convenient position in the list arrangement. read-up.

1. **RICHTEXT**

**RichText** widget is used to create text have have different style in it. read-up.

1. **SAFEAREA**

**SafeArea** is used the Create a page that has margin at its top to avoid device nolch from blocking the top part of your page.

1. **SELECTABLETEXT**

**SelectableText** is used to create text that be copy like in web pages. read-up.

1. **SEMANTICS**

**Semantics** is used to… read-up.

1. **SHADER**

**Shader** are used mostly to add transition color or image to your widget. read-up.

1. **SIZEDBOX**

**SizedBox** is used to constrain the size of your widget. read-up.

1. **SLIDER**

**Slider** is used to create a slide always used to select value by user adjusting the slide. read-up. RangeSlider is used in Slider to create range where your slide will start by default.

1. **SLIVERAPPBAR**

**SliverAppBar** is very useful. read-up.

1. **SLIVERLIST**

**SliverList** is also very useful. read-up.

1. **SNACKBAR**

**SnackBar** is used inside Scaffold to display quick information. read-up.

1. **SPACER**

**Spacer** is used to create flexible space between two widgets. read-up.

1. **STREAMBUILDER**

**StreamBuilder** is used to download or load files with multiple values.

1. **SWITCHLISTTILE**

**SwitchListTile** and it’s Brothers like CheckBoxListTile, RadioListTile are used to … read-up.

1. **TABLE**

**Table** is used to create tables.

1. **TOGGLEBUTTON**

**ToggleButton** is used to … read-up.

1. **TOOLTIP**

**ToolTip** is used to displace some text of description for a particular widget when it is been hovered… read-up.

1. **TRANSFORM**

**Transform** helps in severe animation.

1. **TWEENANIMATIONBUILDER**

**TweenAnimationBuilder** is used… read up.

1. **VALUELISTENABLEBUILDER**

**ValueListenableBuilder** is used to change the value of widgets periodically. it is mostly used in post pages. read\_up.

1. **WRAP**

**Wrap** is used to replace Column and Rows that want it to go to the next line when it is overflowing.

1. DONE!.

**List**

Dismissible(…)

* + 1. **LISTTILE**

ListTile widget is used to create a widget like those in chats,

Example:



ListTile have more useful parameter like, **trailing: icon** is the equal to sign… etc.

* + 1. **LISTVIEW**

**ListView** is used to create a scrollable list. It can be in the horizontal or vertical direction. it has much named constructors and parameter that are useful. Read-up.

* + 1. **LISTWHEELSCROLLVIEW**

**ListWheelScrollView** is an animated ListView mostly used in Post page or media( pic, video… etc) page to make them more beautiful.

* + 1. .

**IMAGE**

1. **IMAGE FILTER:**

**BackdropFIlter** widget can be used with **ImageFilter** to filter widgets below the **BackdropFilter**.

The **BackdropFilter** child widget is not affected by the filter, it is only those widgets below it that are affected.

**BackdropFilter** filter parameter is required, so it must be specified. It can have value of

* filter: ImageFilter.blur(…)
* filter: ImageFilter.compose(…)
* filter: ImageFilter.dilate(…)
* filter: ImageFilter.erode(…)
* filter: ImageFilter.matrix(…)

You can use **Positioned** widget to specify where the filter will affect.   
Example:

    return Scaffold(

      body: Stack(

        children: [

          // Position.fill make the whole Widget in the stack

          //to be blurred because it set

          //top,bottom,left,and right parameter to 0.0

          Positioned.fill(

            // top,bottom,left and right when set, are Used to apply

            //the filter to a specific place.

            child: BackdropFilter(

              filter: ImageFilter.blur(

                sigmaX: 5,

                sigmaY: 5,

              ),

              child: Container(

                color: Colors.blue.withOpacity(0.2),

              ),

            ),

          ),

          //Place Other widgets you want to filter here

        ],

      ),

    );

1. **COlORFILTERED(…):**

**ColorFiltered** is a widget Used to add color to an Image without covering the Image with the Color.

The colorFilter parameter must be set, which can have a value of:

* ColorFilter.linearToSrgbGamma()
* ColorFilter.matrix(…)
* ColorFilter.mode(…)
* ColorFilter. SrgbToLinearGamma()

Example:

child: ColorFiltered(

        colorFilter: ColorFilter.mode(Colors.green, BlendMode.modulate),

        child: Image.asset("url"),

      ),

which results to the below image.



1. **IMAGE**

Image Widget is used to add images to your App from file,network… etc. It has useful named constructor like, like Image.asset(…), .network(…),… etc.

1. .

**FORM**

**CHECKBOXLISTTILE:**

**CheckboxListTile** is a widget used to add checkbox with much designs

**ANIMATION**

**AnimatedBuilder:**

It is used to build a widget that can rotate.

final animation = Tween(begin: 0, end: 2 \* pi).animate(\_animationController);

late final AnimationController \_animationController;

AnimatedBuilder(

  animation: animation,

  child: FlutterLogo(),//This is where you will Add the widget u want to Animate.  
  builder: (context, child) {  
 return Transform.rotate(  
 angle: 90 \* pi,  
 child: child,  
 );  
 },  
),

**AnimatedCrossFade:**

This is used to fade from one widget( image, text, video etc.) to another widget.

Animated builder got you covered when the two interchanging widgets have different size or shape. This is possible through the key property of any widget.

Check **AnimatedSwitcher** below, it’s like it’s better.

Example:

When the widget have the **same** size:

AnimatedCrossFade(

        duration: Duration(milliseconds: 200),

        crossFadeState:

            false ? CrossFadeState.showSecond : CrossFadeState.showFirst,

        firstChild: FirstWidget(),

        secondChild: SecondWidget(),

      ),

When the widgets have **different** sizes:

       AnimatedCrossFade(

        duration: Duration(milliseconds: 200),

        crossFadeState:

            false ? CrossFadeState.showSecond : CrossFadeState.showFirst,

        firstChild: FirstWidget(),

        secondChild: SecondWidget(),

        layoutBuilder: (firstWidget, firstKey, secondWidget, secondKey) {

          return Stack(

            alignment: Alignment.center,

            children: [

              Positioned(

                child: secondWidget,

                top: 0,

                key: secondKey,

              ),

              Positioned(

                child: firstWidget,

                top: 0,

                key: firstKey,

              )

            ],

          );

        },

      ),

**AnimatedIcon:**

There are main **AnimatedIcon** that Flutter provides in **AnimatedIcons**.

AnimatedIcon(

        icon: AnimatedIcons.play\_pause,

        progress: myAnimation,

      ),

**AnimatedList:**

// I will learn more when I need it.

**AnimatedOpacity:**

// Learn when needed

also learn **FadeTransition**

AnimatedOpacity(

        duration: Duration(milliseconds: 200),

        opacity: 0.5,

        child: YourWidget(),

      ),

**AnimatedPadding:**

// Learn when needed

**AnimatedPosition:**

It is mostly used for slides.

AnimatedPositioned(

        duration: Duration(milliseconds: 200),

        bottom: condition?1:10,

        right: condition?10:1,

        child: condition? FirstWidget(): SecondWidget(),

      )

**AnimatedSwitcher:**

It is used to switch between widget. you can specify a transition type from the Transitions( SlideTransition, RotationTransition,…etc) provided by Flutter.

//Try and learn more about it.

Example:

**AnimatedWidget:**

It is an abstract class that helps you to create your own animation.

1. **.**

**PACKAGES**

1. **LOCATION**

**location** package is used to get the client location. read-up.

1. **SQFLITE**

**sqflite** package is used to save data to your local database.

1. **URL\_LAUNCHER**

**url\_launcher** package is used to leave your app to other apps like web browser, phone dial, youtube,… etc.

1. .
2. **.**

**24.**

**UTILITIES AND THEIR FUNCTION.**

|  |  |
| --- | --- |
| WIDGETS | FUNCTIONS AND COMPONENTS |
| MaterialApp() | This is the Top-most widget that is always returned by build(…) method. Although build(…) method can return any widget, like Scaffold(…).  It contains parameters like **title**, **key**, **home**, **route**, **navigatorObservers** etc. you can hover on it to see more. |
| Scaffold(…) | It is the Widget that contains almost all the basic part of your App.  It is usually the widget at MaterialApp()’s home parameter.  It has parameters like **appBar**, **key**, **body**, **drawer**, **floatActionButton** etc. |
| Center(…) | It is used to align it’s child widget to the center of the screen.  It has parameters like **key**, **widthFactor**, **heightFactor**, **child** etc. |
| Container(…) | This widget is used to contain widget that you need to add more features to.  It has parameters like **key**, **color**, **alignment**,**padding**, **decoration**, **constraint**, **transform**, **width, child**, … etc. |
| ListView(…) | It allows you to build a list lazily on demand. |
| AbsorbPointer() | It is used to hold contain any widget that u don’t it to be clickable.Example:  Return AbsorbPointer(  ignoringSemantics: false,  child: OtherWidgets()  ) |
| AlertDialog(…)  Or  CurpertinoAle… | It is used to prompt a user. Example:  AlertDialog(  title: Text(“ Accept”),  content: TextOrImageOrAnimation(“Do you accept”);  action: [  FlatButton(“No”),  FlatButton(“Yes”),  ],  ), |
| Align(…) | This is used to place a widget in a specific place in it’s parent. Example:  Container(  child: Align(  alignment: Alignment.centerRight,  child: Text(“Love”),  ),  ), |
| Builder(…) |  |
| ConstraintBox(…) | It is Used to Create a widget with a spe |
|  |  |

**4.**

**PACKAGES AND THEIR FUNCTIONS**

|  |  |
| --- | --- |
| WIDGETS | FUNCTIONS AND COMPONENTS |
| uses-material-design: true | It will allow you to use more feature of Material, such as their set of predefined [Icons](https://fonts.google.com/icons). |
| english\_words: ^4.0.0 | It contain most used English words. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Flutter's build modes

Contents

* [Debug](https://docs.flutter.dev/testing/build-modes#debug)
* [Release](https://docs.flutter.dev/testing/build-modes#release)
* [Profile](https://docs.flutter.dev/testing/build-modes#profile)

The Flutter tooling supports three modes when compiling your app, and a headless mode for testing. You choose a compilation mode depending on where you are in the development cycle. Are you debugging your code? Do you need profiling information? Are you ready to deploy your app?

A quick summary for when to use which mode is as follows:

* Use [debug](https://docs.flutter.dev/testing/build-modes#debug) mode during development, when you want to use [hot reload](https://docs.flutter.dev/development/tools/hot-reload).
* Use [profile](https://docs.flutter.dev/testing/build-modes#profile) mode when you want to analyze performance.
* Use [release](https://docs.flutter.dev/testing/build-modes#release) mode when you are ready to release your app.

The rest of the page goes into more detail about these modes. For information on headless testing, see the [Flutter wiki](https://github.com/flutter/flutter/wiki/Flutter's-modes).

## Debug

In debug mode, the app is set up for debugging on the physical device, emulator, or simulator.

Debug mode for mobile apps mean that:

* [Assertions](https://dart.dev/guides/language/language-tour#assert) are enabled.
* Service extensions are enabled.
* Compilation is optimized for fast development and run cycles (but not for execution speed, binary size, or deployment).
* Debugging is enabled, and tools supporting source level debugging (such as [DevTools](https://docs.flutter.dev/development/tools/devtools)) can connect to the process.

Debug mode for a web app means that:

* The build is not minified and tree shaking has not been performed.
* The app is compiled with the [dartdevc](https://dart.dev/tools/dartdevc) compiler for easier debugging.

By default, flutter run compiles to debug mode. Your IDE supports this mode. Android Studio, for example, provides a **Run > Debug…** menu option, as well as a green bug icon overlayed with a small triangle on the project page.

**Note:**

* Hot reload works only in debug mode.
* The emulator and simulator execute only in debug mode.
* Application performance can be janky in debug mode. Measure performance in [profile](https://docs.flutter.dev/testing/build-modes#profile) mode on an actual device.

## Release

Use release mode for deploying the app, when you want maximum optimization and minimal footprint size. For mobile, release mode (which is not supported on the simulator or emulator), means that:

* Assertions are disabled.
* Debugging information is stripped out.
* Debugging is disabled.
* Compilation is optimized for fast startup, fast execution, and small package sizes.
* Service extensions are disabled.

Release mode for a web app means that:

* The build is minified and tree shaking has been performed.
* The app is compiled with the [dart2js](https://dart.dev/tools/dart2js) compiler for best performance.

The command flutter run --release compiles to release mode. Your IDE supports this mode. Android Studio, for example, provides a **Run > Run…** menu option, as well as a triangular green run button icon on the project page. You can compile to release mode for a specific target with flutter build <target>. For a list of supported targets, use flutter help build.

For more information, see the docs on releasing [iOS](https://docs.flutter.dev/deployment/ios) and [Android](https://docs.flutter.dev/deployment/android) apps.

## Profile

In profile mode, some debugging ability is maintained—enough to profile your app’s performance. Profile mode is disabled on the emulator and simulator, because their behavior is not representative of real performance. On mobile, profile mode is similar to release mode, with the following differences:

* Some service extensions, such as the one that enables the performance overlay, are enabled.
* Tracing is enabled, and tools supporting source-level debugging (such as [DevTools](https://docs.flutter.dev/development/tools/devtools)) can connect to the process.

Profile mode for a web app means that:

* The build is not minified but tree shaking has been performed.
* The app is compiled with the [dart2js](https://dart.dev/tools/dart2js) compiler.

Your IDE supports this mode. Android Studio, for example, provides a **Run > Profile…** menu option. The command flutter run --profile compiles to profile mode.

**Note:** Use the [DevTools](https://docs.flutter.dev/development/tools/devtools) suite to profile your app’s performance.

For more information on the build modes, see [Flutter’s build modes](https://github.com/flutter/flutter/wiki/Flutter%27s-modes).