

**(Flutter)**

Flutter Lab 1 - Session 2

**Total Time:**

**3 hours**

**Pre-Lab Activities:**

* No Pre-Lab Activity

**Learning Outcomes:**

* Perform the execution, debugging, testing, and profiling of mobile apps in modern IDEs.

**Lab Tasks:**

* + To install and configure the Flutter.
  + To explore the Android Studio functionalities.

**Student Activities:**

* + To install and configure the Flutter.

**Lab Solutions:**

# Lab Task 1: Install and configure Flutter

## System requirements

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

* **Operating Systems**: Windows 7 SP1 or later (64-bit), x86-64 based.
* **Disk Space**: 1.64 GB (does not include disk space for IDE/tools).
* **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 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 or PowerShell.

## Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

[flutter\_windows\_2.5.2-stable.zip](https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_2.5.2-stable.zip)

For other release channels, and older builds, see the [SDK releases](https://flutter.dev/docs/development/tools/sdk/releases) page.

1. Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\Users\<your-user-name>\Documents).

**Warning:** Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges.

If you don’t want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the [Flutter repo](https://github.com/flutter/flutter) on GitHub, and change branches or tags as needed. For example:

C:\src>git clone https://github.com/flutter/flutter.git -b stable

You are now ready to run Flutter commands in the Flutter Console.

### Update your path

If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the PATH environment variable:

* From the Start search bar, enter ‘env’ and select **Edit environment variables for your account**.
* Under **User variables** check if there is an entry called **Path**:
  + If the entry exists, append the full path to flutter\bin using ; as a separator from existing values.
  + If the entry doesn’t exist, create a new user variable named Path with the full path to flutter\bin as its value.

You have to close and reopen any existing console windows for these changes to take effect.

### Run flutter doctor

From a console window that has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:

C:\src\flutter>flutter doctor

This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).

For example:

[-] Android toolchain - develop for Android devices

• Android SDK at D:\Android\sdk

**✗ Android SDK is missing command line tools; download from https://goo.gl/XxQghQ**

• Try re-installing or updating your Android SDK,

visit https://flutter.dev/setup/#android-setup for detailed instructions.

## Create the app

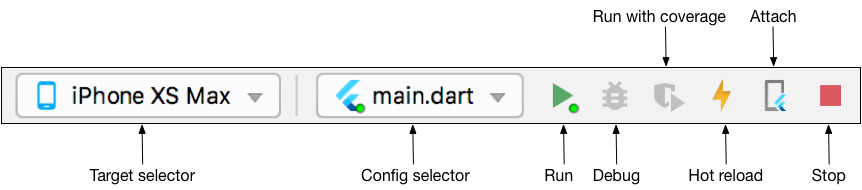
1. Open the IDE and select **Create New Flutter Project**.
2. Select **Flutter Application** as the project type. Then click **Next**.
3. Verify the Flutter SDK path specifies the SDK’s location (select **Install SDK…** if the text field is blank).
4. Enter a project name (for example, myapp). Then click **Next**.
5. Click **Finish**.
6. Wait for Android Studio to install the SDK and create the project.

**Note:** When creating a new Flutter app, some Flutter IDE plugins ask for a company domain name in reverse order, something like com.example. The company domain name and project name are used together as the package name for Android (the Bundle ID for iOS) when the app is released. If you think that the app might be released, it’s better to specify the package name now. The package name can’t be changed once the app is released, so make the name unique.

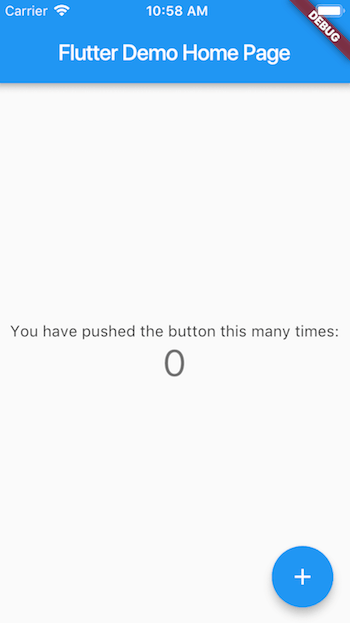
The above commands create a Flutter project directory called myapp that contains a simple demo app that uses [Material Components](https://material.io/guidelines).

**Tip:** The code for your app is in lib/main.dart. For a high-level description of what each code block does, see the comments at the top of that file.

## Run the app

1. Locate the main Android Studio toolbar:  
   
2. In the **target selector**, select an Android device for running the app. If none are listed as available, select **Tools > AVD Manager** and create one there. For details, see [Managing AVDs](https://developer.android.com/studio/run/managing-avds).
3. Click the run icon in the toolbar, or invoke the menu item **Run > Run**.

After the app build completes, you’ll see the starter app on your device.

Starter app

## Try hot reload

Flutter offers a fast development cycle with Stateful Hot Reload, the ability to reload the code of a live running app without restarting or losing app state. Make a change to app source, tell your IDE or command-line tool that you want to hot reload, and see the change in your simulator, emulator, or device.

1. Open lib/main.dart.
2. Change the string

'You have ~~pushed~~ the button this many times'

to

'You have clicked the button this many times'

**Important:** Do not stop your app. Let your app run.

1. Save your changes: invoke **Save All**, or click **Hot Reload**  .

You’ll see the updated string in the running app almost immediately.

## Profile or release runs

**Important:** Do not test the performance of your app with debug and hot reload enabled.

So far you’ve been running your app in debug mode. Debug mode trades performance for useful developer features such as hot reload and step debugging. It’s not unexpected to see slow performance and janky animations in debug mode. Once you are ready to analyze performance or release your app, you’ll want to use Flutter’s “profile” or “release” build modes. For more details, see [Flutter’s build modes](https://flutter.dev/docs/testing/build-modes).

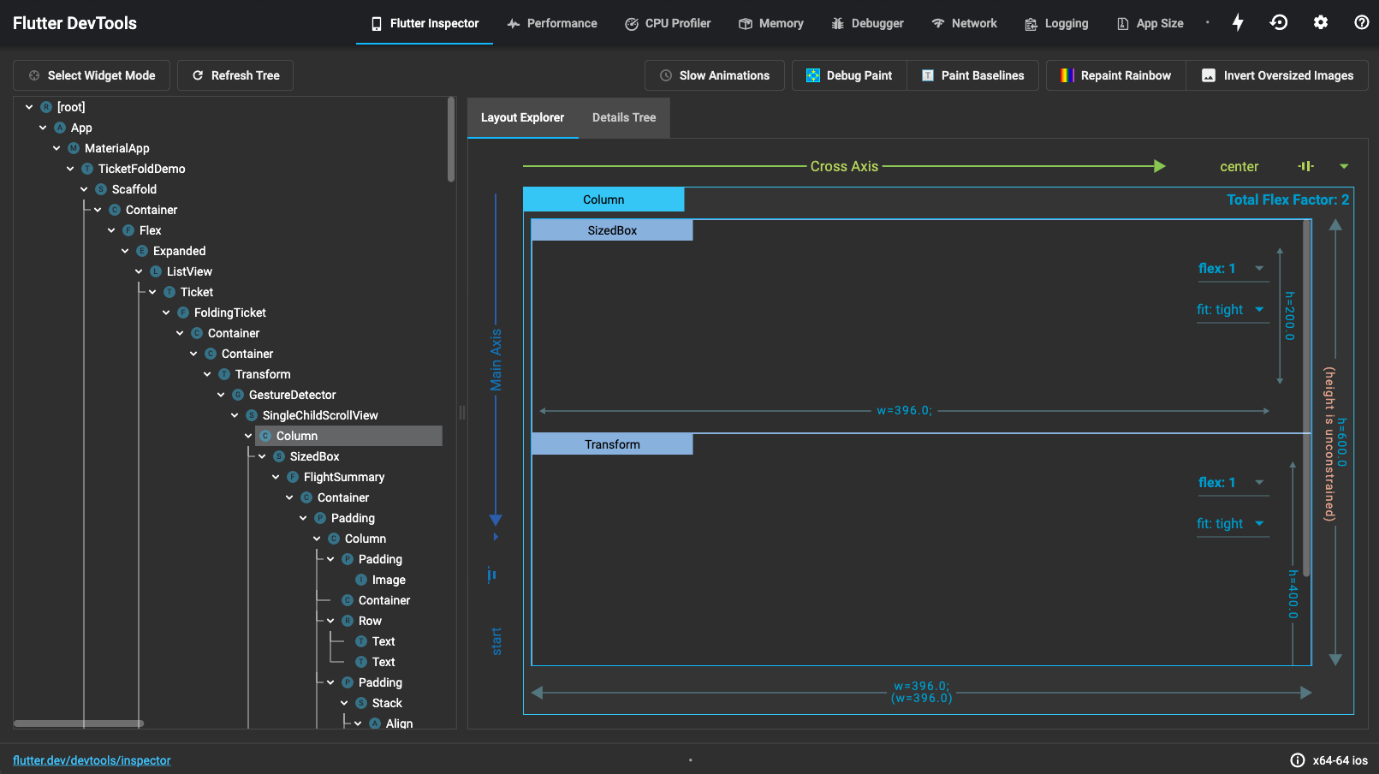
**Important:** If you’re concerned about the package size of your app, see [Measuring your app’s size](https://flutter.dev/docs/perf/app-size).

# 5 - Using the Flutter inspector

## What is it?

The Flutter widget inspector is a powerful tool for visualizing and exploring Flutter widget trees. The Flutter framework uses widgets as the [core building block](https://flutter.dev/docs/development/ui/widgets-intro) for anything from controls (such as text, buttons, and toggles), to layout (such as cantering, padding, rows, and columns). The inspector helps you visualize and explore Flutter widget trees, and can be used for the following:

* understanding existing layouts
* diagnosing layout issues



## Get started

To debug a layout issue, run the app in [debug mode](https://flutter.dev/docs/testing/build-modes#debug) and open the inspector by clicking the **Flutter Inspector** tab on the DevTools toolbar.

**Note:** You can still access the Flutter inspector directly from Android Studio/IntelliJ, but you might prefer the more spacious view when running it from DevTools in a browser.

### Debugging layout issues visually

The following is a guide to the features available in the inspector’s toolbar. When space is limited, the icon is used as the visual version of the label.

Select widget mode icon **Select widget mode**

Enable this button in order to select a widget on the device to inspect it. For more information, see [Inspecting a widget](https://flutter.dev/docs/development/tools/devtools/inspector#inspecting-a-widget).

Refresh tree icon **Refresh tree**

Reload the current widget info.

Slow animations icon [**Slow animations**](https://flutter.dev/docs/development/tools/devtools/inspector#slow-animations)

Run animations 5 times slower to help fine-tune them.

Show guidelines mode icon [**Show guidelines**](https://flutter.dev/docs/development/tools/devtools/inspector#show-guidelines)

Overlay guidelines to assist with fixing layout issues.

Show baselines icon [**Show baselines**](https://flutter.dev/docs/development/tools/devtools/inspector#show-baselines)

Show baselines, which are used for aligning text. Can be useful for checking if text is aligned.

Highlight repaints icon [**Highlight repaints**](https://flutter.dev/docs/development/tools/devtools/inspector#highlight-repaints)

Show borders that change color when elements repaint. Useful for finding unnecessary repaints.

Highlight oversized images icon [**Highlight oversized images**](https://flutter.dev/docs/development/tools/devtools/inspector#highlight-oversized-images)

Highlights images that are using too much memory by inverting colors and flipping them.

## Inspecting a widget

You can browse the interactive widget tree to view nearby widgets and see their field values.

To locate individual UI elements in the widget tree, click the **Select Widget Mode** button in the toolbar. This puts the app on the device into a “widget select” mode. Click any widget in the app’s UI; this selects the widget on the app’s screen, and scrolls the widget tree to the corresponding node. Toggle the **Select Widget Mode** button again to exit widget select mode.

When debugging layout issues, the key fields to look at are the size and constraints fields. The constraints flow down the tree, and the sizes flow back up. For more information on how this works, see [Understanding constraints](https://flutter.dev/docs/development/ui/layout/constraints).