Contributing to Flutter

*See also:*[*Flutter's code of conduct*](https://flutter.io/design-principles/#code-of-conduct)

Things you will need

* Linux, Mac OS X, or Windows
* git (used for source version control).
* An IDE. We recommend [IntelliJ with the Flutter plugin](https://flutter.io/intellij-ide/).
* An ssh client (used to authenticate with GitHub).
* Python (used by some of our tools).
* The Android platform tools (see [Issue #55](https://github.com/flutter/flutter/issues/55) about downloading the Android platform tools automatically). *If you're also working on the Flutter engine, you can use the copy of the Android platform tools in .../engine/src/third\_party/android\_tools/sdk/platform-tools.*
  + Mac: brew install android-platform-tools
  + Linux: sudo apt-get install android-tools-adb

Getting the code and configuring your environment

* Ensure all the dependencies described in the previous section, in particular git, ssh, and python are installed. Ensure that adb (from the Android platform tools) is in your path (e.g., that which adb prints sensible output).
* Fork https://github.com/flutter/flutter into your own GitHub account. If you already have a fork, and are now installing a development environment on a new machine, make sure you‘ve updated your fork so that you don’t use stale configuration options from long ago.
* If you haven‘t configured your machine with an SSH key that’s known to github then follow the directions here: <https://help.github.com/articles/generating-ssh-keys/>.
* git clone git@github.com:<your\_name\_here>/flutter.git
* cd flutter
* git remote add upstream git@github.com:flutter/flutter.git (So that you fetch from the master repository, not your clone, when running git fetch et al.)
* Add this repository's bin directory to your path. That will let you use the flutter command in this directory more easily.
* Run flutter update-packages This will fetch all the Dart packages that Flutter depends on. You can replicate what this script does by running pub get in each directory that contains a pubspec.yaml file.
* If you plan on using IntelliJ as your IDE, then also run flutter ide-config --overwrite to create all of the IntelliJ configuration files so you can open the main flutter directory as a project and run examples from within the IDE.

Running the examples

To run an example, switch to that example's directory, and use flutter run. Make sure you have an emulator running, or a device connected over USB and debugging enabled on that device.

* cd examples/hello\_world
* flutter run

You can also specify a particular Dart file to run if you want to run an example that doesn't have a lib/main.dart file using the -t command-line option. For example, to run the widgets/spinning\_square.dart example in the [examples/layers](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/examples/layers) directory on a connected Android device, from that directory you would run: flutter run -t widgets/spinning\_square.dart

When running code from the examples directory, any changes you make to the example code, as well as any changes to Dart code in the [packages/flutter](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/packages/flutter) directory and subdirectories, will automatically be picked when you relaunch the app. You can do the same for your own code by mimicking the pubspec.yaml files in the examples subdirectories.

Running the analyzer

When editing Flutter code, it's important to check the code with the analyzer. There are two main ways to run it. In either case you will want to run flutter update-packages first, or you will get bogus error messages about core classes like Offset from dart:ui.

For a one-off, use flutter analyze --flutter-repo. This uses the analysis\_options\_repo.yaml file at the root of the repository for its configuration.

For continuous analysis, use flutter analyze --flutter-repo --watch. This uses normal analysis\_options.yaml files, and they can differ from package to package.

If you want to see how many members are missing dartdocs, you should use the first option, providing the additional command --dartdocs.

If you omit the --flutter-repo option you may end up in a confusing state because that will assume you want to check a single package and the flutter repository has several packages.

Running the tests

To automatically find all files named \_test.dart inside a package's test/ subdirectory, and run them inside the flutter shell as a test, use the flutter test command, e.g:

* cd examples/stocks
* flutter test

Individual tests can also be run directly, e.g. flutter test lib/my\_app\_test.dart

Flutter tests use [package:flutter\_test](https://github.com/flutter/flutter/tree/master/packages/flutter_test) which provides flutter-specific extensions on top of [package:test](https://pub.dartlang.org/packages/test).

flutter test runs tests inside the flutter shell. To debug tests in Observatory, use the --start-paused option to start the test in a paused state and wait for connection from a debugger. This option lets you set breakpoints before the test runs.

To run all the tests for the entire Flutter repository, the same way that Travis runs them, run dart dev/bots/test.dart.

If you've built [your own flutter engine](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/CONTRIBUTING.md#working-on-the-engine-and-the-framework-at-the-same-time), you can pass --local-engine to change what flutter shell flutter test uses. For example, if you built an engine in the out/host\_debug\_unopt directory, you can pass --local-engine=host\_debug\_unopt to run the tests in that engine.

Flutter tests are headless, you won't see any UI. You can use print to generate console output or you can interact with the DartVM via observatory at <http://localhost:8181/>.

Adding a test

To add a test to the Flutter package, create a file whose name ends with \_test.dart in the packages/flutter/test directory. The test should have a main function and use the test package.

Working with flutter tools

The flutter tool itself is built when you run flutter for the first time and each time you run flutter upgrade. If you want to alter and re-test the tool's behavior itself, locally commit your tool changes in git and the tool will be rebuilt from Dart sources in packages/flutter\_tools the next time you run flutter.

Alternatively, delete the bin/cache/flutter\_tools.snapshot file. Doing so will force a rebuild of the tool from your local sources the next time you run flutter.

flutter\_tools' tests run inside the Dart command line VM rather than in the flutter shell. To run the tests, ensure that no devices are connected, then navigate to flutter\_tools and execute:

../../bin/cache/dart-sdk/bin/pub run test -j1

The pre-built flutter tool runs in release mode with the observatory off by default. To enable debugging mode and the observatory on the flutter tool, uncomment the FLUTTER\_TOOL\_ARGS line in the bin/flutter shell script.

Contributing code

We gladly accept contributions via GitHub pull requests.

Please peruse our [style guides](https://github.com/flutter/flutter/wiki/Style-guide-for-Flutter-repo) and [design principles](https://flutter.io/design-principles/) before working on anything non-trivial. These guidelines are intended to keep the code consistent and avoid common pitfalls.

To start working on a patch:

* git fetch upstream
* git checkout upstream/master -b name\_of\_your\_branch
* Hack away.
* git commit -a -m "<your informative commit message>"
* git push origin name\_of\_your\_branch

To send us a pull request:

* git pull-request (if you are using [Hub](http://github.com/github/hub/)) or go to https://github.com/flutter/flutter and click the “Compare & pull request” button

Please make sure all your checkins have detailed commit messages explaining the patch.

Once you've gotten an LGTM from a project maintainer and once your PR has received the green light from all our automated testing (Travis, Appveyor, etc), and once the tree is green (see the [design principles](https://flutter.io/design-principles/) document for more details), submit your changes to the master branch using one of the following methods:

* Wait for one of the project maintainers to submit it for you.
* Click the green “Merge pull request” button on the GitHub UI of your pull request (requires commit access)

You must complete the [Contributor License Agreement](https://cla.developers.google.com/clas). You can do this online, and it only takes a minute. If you‘ve never submitted code before, you must add your (or your organization’s) name and contact info to the [AUTHORS](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/AUTHORS) file.

We grant commit access to people who have gained our trust and demonstrated a commitment to Flutter.

Tools for tracking and improving test coverage

We strive for a high degree of test coverage for the Flutter framework. We use Coveralls to [track our test coverage](https://coveralls.io/github/flutter/flutter?branch=master). You can download our current coverage data from cloud storage and visualize it in Atom as follows:

* Install [Atom](https://atom.io/).
* Install the [lcov-info](https://atom.io/packages/lcov-info) package for Atom.
* Open the packages/flutter folder in Atom.
* Open a Dart file in the lib directory an type Ctrl+Alt+C to bring up the coverage data.

If you don't see any coverage data, check that you have an lcov.info file in the packages/flutter/coverage directory. It should have been downloaded by the flutter update-packages command you ran previously.

If you want to iterate quickly on improving test coverage, consider using this workflow:

* Open a file and observe that some line is untested.
* Write a test that exercises that line.
* Run flutter test --merge-coverage path/to/your/test\_test.dart.
* After the test passes, observe that the line is now tested.

This workflow merges the coverage data from this test run with the base coverage data downloaded by flutter update-packages.

See [issue 4719](https://github.com/flutter/flutter/issues/4719) for ideas about how to improve this workflow.

Working on the engine and the framework at the same time

You can work both with this repository (flutter.git) and the Flutter [engine repository](https://github.com/flutter/engine) at the same time using the following steps.

1. Follow the instructions above for creating a working copy of this repository.
2. Follow the [contributing instructions](https://github.com/flutter/engine/blob/master/CONTRIBUTING.md) for the engine repository to create a working copy of the engine. When you create the .gclient file for the engine, be sure to create it in a directory named engine that is a sibling of the directory in which you cloned this repository. For example, if you cloned this repository into the /foo/bar/flutter directory, you should create the .gclient file in the /foo/bar/engine directory. The actual code from the engine repository will end up in /foo/bar/engine/src because gclient creates a src directory underneath the directory that contains the .gclient file.
3. To run tests on your host machine, build one of the host configurations (e.g., out/host\_debug\_unopt). To run examples on Android, build one of the Android configurations (e.g., out/android\_debug\_unopt). When running on the device with --preview-dart-2 flag you will still need to build corresponding host configuration (e.g., out/host\_debug\_unopt if you are using out/android\_debug\_unopt, out/host\_release if you use out/android\_release). Host configuration provides standalone dart sdk for the engine, that is used to run engine dart scripts on the host.

You should now be able to run the tests against your locally built engine using the flutter test --local-engine=host\_debug\_unopt command. To run one of the examples on your device using your locally built engine, use the --local-engine=android\_debug\_unopt option to the flutter tool:

* flutter run --local-engine=android\_debug\_unopt

Making a breaking change to the engine

If you make a breaking change to the engine, you'll need to land your change in a few steps:

1. Land your change in the engine repository.
2. Publish a new version of the engine that contains your change. See the engine's [release process](https://github.com/flutter/engine/wiki/Release-process) for instructions about how to publish a new version of the engine. Publishing a new version is important in order to not break folks using prebuilt binaries in their workflow (e.g., our customers).
3. Land a change that update our dependency on the sky\_engine and sky\_services packages to point to the new version of the engine that you just published. These dependencies are defined by [packages/flutter/pubspec.yaml](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/packages/flutter/pubspec.yaml). After changing the pubspec.yaml file, you'll need to run ./dev/update\_packages.dart to update all the packages in this repository to see the new dependency. As part of landing this change, you should make whatever other changes are needed in this repository to account for your breaking change.

API docs for master branch

To view the API docs for the master branch, visit <https://master-docs-flutter-io.firebaseapp.com/>.

Those docs should be updated after a successful CI build of Flutter's master branch.

(Looking for the API docs for our releases? Please visit [https://docs.flutter.io](https://docs.flutter.io/).)

Build infrastructure

We build and test Flutter on:

* Travis ([details](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/.travis.yml))
* AppVeyor ([details](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/appveyor.yml))
* Chromebots (a.k.a. “recipes”, [details](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/dev/bots/README.md))
* Devicelab (a.k.a. “cocoon”, [details](https://chromium.googlesource.com/external/github.com/flutter/flutter/+/refs/tags/v0.2.3/dev/devicelab/README.md))

Powered by [Gitiles](https://gerrit.googlesource.com/gitiles/)| [Privacy](https://policies.google.com/privacy)| [Terms](https://policies.google.com/terms)