Flutter Tools

This section of the Flutter repository contains the command line developer tools for building Flutter applications.

Working on Flutter Tools

Be sure to follow the instructions on CONTRIBUTING.md to set up your development environment. Further, familiarize yourself with the style guide, which we follow.

Setting up

First, ensure that the Dart SDK and other necessary artifacts are available by invoking the Flutter Tools wrapper script. In this directory run:

```
$ flutter --version
```

Running the Tool

To run Flutter Tools from source, in this directory run:

```
$ dart bin/flutter_tools.dart
```

followed by command-line arguments, as usual.

Running the analyzer

To run the analyzer on Flutter Tools, in this directory run:

```
$ flutter analyze
```

Writing tests

As with other parts of the Flutter repository, all changes in behavior must be tested. Tests live under the `test/` subdirectory.

- Hermetic unit tests of tool internals go under `test/general.shard` and must run in significantly less than two seconds.
- Tests of tool commands go under `test/commands.shard`. Hermetic tests go under its `hermetic/` subdirectory. Non-hermetic tests go under its `permeable` sub-directory. Avoid adding tests here and prefer writing either a unit test or a full integration test.
- Integration tests (e.g. tests that run the tool in a subprocess) go under `test/integration.shard`.
- Slow web-related tests go in the `test/web.shard` directory.

In general, the tests for the code in a file called `file.dart` should go in a file called `file_test.dart` in the subdirectory that matches the behavior of the test.

The `dart_test.yaml` file configures the timeout for these tests to be 15 minutes. The `test.dart` script that is used in CI overrides this to two seconds for the `test/general.shard` directory, to catch behaviour that is

unexpectedly slow.

Please avoid setting any other timeouts.

Using local engine builds in integration tests

The integration tests can be configured to use a specific local engine variant by setting the `FLUTTER_LOCAL_ENGINE` environment variable to the name of the local engine (e.g. "android_debug_unopt"). If the local engine build requires a source path, this can be provided by setting the `FLUTTER_LOCAL_ENGINE_SRC_PATH` environment variable. This second variable is not necessary if the `flutter` and `engine` checkouts are in adjacent directories.

```
export FLUTTER_LOCAL_ENGINE=android_debug_unopt
flutter test test/integration.shard/some_test_case
```

Running the tests

To run all of the unit tests:

```
$ flutter test test/general.shard
```

The tests in `test/integration.shard` are slower to run than the tests in `test/general.shard`. Depending on your development computer, you might want to limit concurrency. Generally it is easier to run these on CI, or to manually verify the behavior you are changing instead of running the test.

The integration tests also require the `FLUTTER_ROOT` environment variable to be set. The full invocation to run everything might therefore look something like:

```
$ export FLUTTER_ROOT=~/path/to/flutter-sdk
$ flutter test --concurrency 1
```

This may take some time (on the order of an hour). The unit tests alone take much less time (on the order of a minute).

You can run the tests in a specific file, e.g.:

```
$ flutter test test/general.shard/utils_test.dart
```

Forcing snapshot regeneration

To force the Flutter Tools snapshot to be regenerated, delete the following files:

```
$ rm ../../bin/cache/flutter_tools.stamp ../../bin/cache/flutter_tools.snapshot
```