integration_test

This package enables self-driving testing of Flutter code on devices and emulators. It adapts flutter_test results into a format that is compatible with `flutter drive` and native Android instrumentation testing.

Usage

Add a dependency on the `integration_test` and `flutter_test` package in the `dev_dependencies` section of `pubspec.yaml`. For plugins, do this in the `pubspec.yaml` of the example app:

```
integration_test:
  sdk: flutter
```

Create a `integration_test/` directory for your package. In this directory, create a `<name>_test.dart`, using the following as a starting point to make assertions.

```
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';

void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  testWidgets("failing test example", (WidgetTester tester) async {
    expect(2 + 2, equals(5));
  });
}
```

Driver Entrypoint

An accompanying driver script will be needed that can be shared across all integration tests. Create a file named `integration_test.dart` in the `test_driver/` directory with the following contents:

```
import 'package:integration_test/integration_test_driver.dart';

Future<void> main() => integrationDriver();
```

You can also use different driver scripts to customize the behavior of the app under test. For example, `FlutterDriver` can also be parameterized with different options. See the extended driver for an example.

Package Structure

Your package should have a structure that looks like this:

```
lib/
```

```
  ...
integration_test/
  foo_test.dart
  bar_test.dart
test/
  # Other unit tests go here.
test_driver/
  integration_test.dart
```

Example

Using Flutter Driver to Run Tests

These tests can be launched with the `flutter drive` command.

To run the `integration_test/foo_test.dart` test with the `test_driver/integration_test.dart` driver, use the following command:

```
flutter drive \
  --driver=test_driver/integration_test.dart \
  --target=integration_test/foo_test.dart
```

Web

Make sure you have enabled web support then download and run the web driver in another process.

Use following command to execute the tests:

```
flutter drive \
  --driver=test_driver/integration_test.dart \
  --target=integration_test/foo_test.dart \
  -d web-server
```

Screenshots

You can use `integration_test` to take screenshots of the UI rendered on the mobile device or Web browser at a specific time during the test.

This feature is currently supported on Android, iOS, and Web.

Android and iOS integration_test/screenshot_test.dart

```
void main() {
  final binding = IntegrationTestWidgetsFlutterBinding.ensureInitialized()
      as IntegrationTestWidgetsFlutterBinding;

  testWidgets('screenshot', (WidgetTester tester) async {
    // Build the app.
    app.main();
```

```
    // This is required prior to taking the screenshot (Android only).
    await binding.convertFlutterSurfaceToImage();

    // Trigger a frame.
    await tester.pumpAndSettle();
    await binding.takeScreenshot('screenshot-1');
  });
}
```

You can use a driver script to pull in the screenshot from the device. This way, you can store the images locally on your computer. On iOS, the screenshot will also be available in Xcode test results.

test_driver/integration_test.dart

```
import 'dart:io';
import 'package:integration_test/integration_test_driver_extended.dart';

Future<void> main() async {
  await integrationDriver(
    onScreenshot: (String screenshotName, List<int> screenshotBytes) async {
      final File image = File('$screenshotName.png');
      image.writeAsBytesSync(screenshotBytes);
      // Return false if the screenshot is invalid.
      return true;
    },
  );
}
```

Web integration_test/screenshot_test.dart

```
void main() {
  final binding = IntegrationTestWidgetsFlutterBinding.ensureInitialized()
      as IntegrationTestWidgetsFlutterBinding;

  testWidgets('screenshot', (WidgetTester tester) async {
    // Build the app.
    app.main();

    // Trigger a frame.
    await tester.pumpAndSettle();
    await binding.takeScreenshot('screenshot-1');
  });
}
```

Android Device Testing

Create an instrumentation test file in your application's **android/app/src/androidTest/java/com/example/** directory (replacing com, example, and myapp with values from your app's package name). You can name this test file `MainActivityTest.java` or another name of your choice.

```
package com.example.myapp;

import androidx.test.rule.ActivityTestRule;
import dev.flutter.plugins.integration_test.FlutterTestRunner;
import org.junit.Rule;
import org.junit.runner.RunWith;

@RunWith(FlutterTestRunner.class)
public class MainActivityTest {
  @Rule
  public ActivityTestRule<MainActivity> rule = new ActivityTestRule<>(MainActivity.class, tr
}
```

Update your application's **myapp/android/app/build.gradle** to make sure it uses androidx's version of `AndroidJUnitRunner` and has androidx libraries as a dependency.

```
android {
  ...
  defaultConfig {
    ...
    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
  }
}

dependencies {
    testImplementation 'junit:junit:4.12'

    // https://developer.android.com/jetpack/androidx/releases/test/#1.2.0
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

To run `integration_test/foo_test.dart` on a local Android device (emulated or physical):

```
./gradlew app:connectedAndroidTest -Ptarget=`pwd`/../integration_test/foo_test.dart
```

Firebase Test Lab

If this is your first time testing with Firebase Test Lab, you'll need to follow the guides in the Firebase test lab documentation to set up a project.

To run a test on Android devices using Firebase Test Lab, use gradle commands to build an instrumentation test for Android, after creating `androidTest` as suggested in the last section.

```
pushd android
# flutter build generates files in android/ for building the app
flutter build apk
./gradlew app:assembleAndroidTest
./gradlew app:assembleDebug -Ptarget=<path_to_test>.dart
popd
```

Upload the build apks Firebase Test Lab, making sure to replace , , , and with your values.

```
gcloud auth activate-service-account --key-file=<PATH_TO_KEY_FILE>
gcloud --quiet config set project <PROJECT_NAME>
gcloud firebase test android run --type instrumentation \
  --app build/app/outputs/apk/debug/app-debug.apk \
  --test build/app/outputs/apk/androidTest/debug/app-debug-androidTest.apk\
  --timeout 2m \
  --results-bucket=<RESULTS_BUCKET> \
  --results-dir=<RESULTS_DIRECTORY>
```

You can pass additional parameters on the command line, such as the devices you want to test on. See gcloud firebase test android run.

iOS Device Testing

Open `ios/Runner.xcworkspace` in Xcode. Create a test target if you do not already have one via `File > New > Target...` and select `Unit Testing Bundle`. Change the `Product Name` to `RunnerTests`. Make sure `Target to be Tested` is set to `Runner` and language is set to `Objective-C`. Select `Finish`. Make sure that the **iOS Deployment Target** of `RunnerTests` within the **Build Settings** section is the same as `Runner`.

Add the new test target to `ios/Podfile` by embedding in the existing `Runner` target.

```
target 'Runner' do
  # Do not change existing lines.
  ...

  target 'RunnerTests' do
    inherit! :search_paths
  end
end
```

To build `integration_test/foo_test.dart` from the command line, run:

```
flutter build ios --config-only integration_test/foo_test.dart
```

In Xcode, add a test file called `RunnerTests.m` (or any name of your choice) to the new target and replace the file:

```
@import XCTest;
@import integration_test;

INTEGRATION_TEST_IOS_RUNNER(RunnerTests)
```

Run `Product > Test` to run the integration tests on your selected device.

To deploy it to Firebase Test Lab you can follow these steps:

Execute this script at the root of your Flutter app:

```
output="../build/ios_integ"
product="build/ios_integ/Build/Products"
dev_target="14.3"

# Pass --simulator if building for the simulator.
flutter build ios integration_test/foo_test.dart --release

pushd ios
xcodebuild -workspace Runner.xcworkspace -scheme Runner -config Flutter/Release.xcconfig -de
popd

pushd $product
zip -r "ios_tests.zip" "Release-iphoneos" "Runner_iphoneos$dev_target-arm64.xctestrun"
popd
```

You can verify locally that your tests are successful by running the following command:

```
xcodebuild test-without-building -xctestrun "build/ios_integ/Build/Products/Runner_iphoneos1
```

Once everything is ok, you can upload the resulting zip to Firebase Test Lab (change the model with your values):

```
gcloud firebase test ios run --test "build/ios_integ/Build/Products/ios_tests.zip" --device
```