Testing Flutter (iOS, Android, web, desktop) Applications

OWenateyou'll learn Prerequisites ApplicatorHow to define tests using Flutter ImplementRundes test and push the test report to Xray
○●UValidate that the test results are available in Jira
 Widget Test
Integration Test
Integrating with Xray
 API Source-code for this tutorialication JUnit XML results codejisavpilable in GitHub
 Tips References

Overview

Flutter is an open source framework by Google used for building natively compiled, multi-platform applications from a single codebase.

Flutter transforms the app development process. Build, test, and deploy mobile, web, desktop, and embedded apps from a single codebase.

Prerequisites

For this example we will use Flutter, that allows the build, test and deploy of mobile, web and desktop applications. We will focus in the testing part of the toolkit.

We use the pre-build demo Flutter counter application, that is available in the code and provided by Flutter, and will define unit tests, widget tests and integrations tests to validate the application.

What you need:

- Flutter installed in your machine
- If you want to run in an iOS emulator you will need the emulator and XCode installed
- If you want to run in an Android emulator you will need to install Android Studio also

Application overview

For the purpose of this tutorial we are using the Flutter counter application that consists in application that will count (and show in the screen) how many times a user pushes the "plus" button.



We only made one change in the generated code that is to include a *Counter* class that will be in charge of the increment.

```
main.dart
import 'package:flutter/material.dart';
import 'package:my_app/counter.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  \ensuremath{{\prime}}\xspace // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        11
        // Try running your application with "flutter run". You'll see the
         \ensuremath{{\prime}}\xspace // application has a blue toolbar. Then, without quitting the app,
trv
         // changing the primarySwatch below to Colors.green and then invoke
         // "hot reload" (press "r" in the console where you ran "flutter
run".
         // or simply save your changes to "hot reload" in a Flutter IDE).
         // Notice that the counter didn't reset back to zero; the
application
         // is not restarted.
         primarySwatch: Colors.blue,
         \ensuremath{{\prime}}\xspace // This makes the visual density adapt to the platform that you run
         // the app on. For desktop platforms, the controls will be smaller
and
         \ensuremath{{\prime}}\xspace ( closer together (more dense) than on mobile platforms.
         visualDensity: VisualDensity.adaptivePlatformDensity,
```

```
),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
class MyHomePage extends StatefulWidget {
 MyHomePage({Key key, this.title}) : super(key: key);
 // This widget is the home page of your application. It is stateful,
meaning
  // that it has a State object (defined below) that contains fields that
affect
  // how it looks.
  // This class is the configuration for the state. It holds the values
(in this
  // case the title) provided by the parent (in this case the App widget)
and
 // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".
 final String title;
  @override
  _MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
 Counter _ counter = new Counter();
 void _incrementCounter() {
    setState(() {
      \ensuremath{{\prime}}\xspace // This call to setState tells the Flutter framework that something
has
      // changed in this State, which causes it to rerun the build method
below
      \ensuremath{//} so that the display can reflect the updated values. If we changed
      // _counter without calling setState(), then the build method would
not be
      // called again, and so nothing would appear to happen.
      _counter.increment();
   });
  }
  @override
  Widget build(BuildContext context) {
    // This method is rerun every time setState is called, for instance as
done
    // by the _incrementCounter method above.
    11
    // The Flutter framework has been optimized to make rerunning build
methods
   // fast, so that you can just rebuild anything that needs updating
rather
    // than having to individually change instances of widgets.
   return Scaffold(
      appBar: AppBar(
        // Here we take the value from the MyHomePage object that was
created by
        // the App.build method, and use it to set our appbar title.
        title: Text(widget.title),
      ),
      body: Center(
       // Center is a layout widget. It takes a single child and
positions it
        // in the middle of the parent.
        child: Column(
          // Column is also a layout widget. It takes a list of children
and
          // arranges them vertically. By default, it sizes itself to fit
```

```
its
           // children horizontally, and tries to be as tall as its parent.
          11
          // Invoke "debug painting" (press "p" in the console, choose the
           // "Toggle Debug Paint" action from the Flutter Inspector in
Android
          // Studio, or the "Toggle Debug Paint" command in Visual Studio
Code)
          \ensuremath{{\prime}}\xspace to see the wireframe for each widget.
          11
          // Column has various properties to control how it sizes itself
and
          // how it positions its children. Here we use mainAxisAlignment
to
          // center the children vertically; the main axis here is the
vertical
          // axis because Columns are vertical (the cross axis would be
          // horizontal).
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
               'You have pushed the button this many times:',
            ).
            Text(
               _counter.getValue().toString(),
               \ensuremath{{\prime}}\xspace // Provide a Key to this specific Text widget. This allows
               // identifying the widget from inside the test suite,
               \ensuremath{{\prime}}\xspace // and reading the text.
              key: Key('counter'),
              style: Theme.of(context).textTheme.headline4,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        // Provide a Key to this button. This allows finding this
        // specific button inside the test suite, and tapping it.
        key: Key('increment'),
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer for build
methods.
    );
  }
}
```

The Counter class will have an increment (), decrement () and getValue() methods available:

counter.dart
<pre>class Counter { int value = 0;</pre>
<pre>void increment() => value++;</pre>
<pre>void decrement() => value;</pre>
<pre>int getValue() => value; }</pre>

Implementing tests

Flutter allows several types of tests, such as:

- Unit tests, that validate a single function, method or class.
- Widget tests, that validate a single widget (referred as component tests in other UI frameworks).
- Integration tests, that validate a complete application or a large part of an application.

If you have doubts regarding Flutter and Flutter tests please check the documentation.

For the purpose of this tutorial we have defined one test of each type to demonstrate how we can integrate those with Xray.

Unit Test

Unit tests are the closest ones to the code and they validate a unit of code, in this case a method: *Count* er.Increment()

These tests are executed with the build and do not require the application to be running.

```
unit_test.dart
// Import the test package and Counter class
import 'package:my_app/counter.dart';
import 'package:test/test.dart';
void main() {
   test('Counter value should be incremented', () {
    final counter = Counter();
    counter.increment();
   expect(counter.getValue(), 1);
  });
}
```

We are calling the method directly and validating that the value was incremented.

Once the code is done we execute it using the following command:

flutter test test/unit_test.dart

In order to integrate with Xray we want to extract the Junit report, for that flutter needs to use the junitr eport Dart package. We need to specify it in the execution command that we are going to generate a Junit report, the execution command will be:

```
flutter pub get junitreport
export PATH="$PATH":"$HOME/.pub-cache/bin"
flutter pub global activate junitreport
junitReportFile="./junit-unit-report.xml"
flutter test --machine test/unit_test.dart | tojunit --output
$junitReportFile
```

The first three lines are there to install and activate the package, the last two lines will execute the tests and generate the "junit-unit-report.xml".

In the report we will find the result of the tests.

junit-unit-report.xml

```
<?rml version="1.0" encoding="UTF-8"?>
<testsuites>
<testsuite errors="0" failures="0" tests="1" skipped="0" name=".Users.
cristianocunha.Documents.Projects.flutter.SF_flutter_demo.my_app.test.
unit" timestamp="2022-10-12T08:25:01">
        <properties>
            <properties>
            <property name="platform" value="vm"/>
            </properties>
            <testcase classname=".Users.cristianocunha.Documents.Projects.flutter.
SF_flutter_demo.my_app.test.unit" name="Counter value should be
incremented" time="0.027"/>
        </testsuite>
</testsuite>
```

Widget Test

The next tests we have defined are the widget tests, tests that will validate a widget (component), for that we have defined the following test:

widget_test.dart

```
// This is a basic Flutter widget test.
11
// To perform an interaction with a widget in your test, use the
WidgetTester
// utility that Flutter provides. For example, you can send tap and scroll
// gestures. You can also use WidgetTester to find child widgets in the
widget
// tree, read text, and verify that the values of widget properties are
correct.
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:my_app/main.dart';
void main() {
  testWidgets('Counter increments smoke test', (WidgetTester tester) async
{
    // Build our app and trigger a frame.
    await tester.pumpWidget(MyApp());
    // Verify that our counter starts at 0.
    expect(find.text('0'), findsOneWidget);
    expect(find.text('1'), findsNothing);
    // Tap the '+' icon and trigger a frame.
    await tester.tap(find.byIcon(Icons.add));
    await tester.pump();
    // Verify that our counter has incremented.
    expect(find.text('0'), findsNothing);
    expect(find.text('1'), findsOneWidget);
  });
}
```

Flutter have available special classes to assist on the validation of widgets, the testWidgets() function allows to define a widget test and creates WidgetTester to work with.

We use the pumpWidget() method of the WidgetTester to build and render our widget (MyApp), next we validate that the application will start up showing 0 in the counter.

We then tap in the plus icon and validate that the counter have now 1.

The widget tests loads a specific widget and, usually, do not require the application to be executed, if your widget depends on other widgets or resources they must be initialized also. In our case our counter application is simple enough that does no have any dependency.

Once the code is ready we execute the tests using the following command:

```
junitReportFile="./junit-widget-report.xml"
flutter test --machine test/widget_test.dart | tojunit --output
$junitReportFile
```

The JUnit report is generated and look like the below one.

Integration Test

The next level of tests are the integration tests and the way we will execute those tests will vary depending on the platform you are testing against. The goal of these tests is to verify that all the widgets and services being tested together work as expected.

As we want to target an Android device we have also installed Android Studio so that we have emulators available. Before executing the Integration Tests we need to start the emulator, please check Flutter documentation on Integration Tests to have more details about this.

Our Integration Test will initialize IntegrationTestWidgetsFlutterBinding, that is a singleton service that executes tests on a physical device, and use WidgetTester to interact and test widgets. This test will follow the same approach as the Widget Tests but now we are executing the application in a device and run the tests against it.

integration_test.dart

```
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:my_app/main.dart' as app;
void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();
  group('end-to-end test', () {
    testWidgets('tap on the floating action button, verify counter',
        (tester) async {
      app.main();
     await tester.pumpAndSettle();
      // Verify the counter starts at 0.
      expect(find.text('0'), findsOneWidget);
      // Finds the floating action button to tap on.
      final Finder fab = find.byTooltip('Increment');
      // Emulate a tap on the floating action button.
      await tester.tap(fab);
      // Trigger a frame.
      await tester.pumpAndSettle();
      // Verify the counter increments by 1.
      expect(find.text('1'), findsOneWidget);
    });
  });
}
```

To execute the test run a command that will execute the application on the target device and perform the tests against it.

```
junitReportFile="./junit-integration-report.xml"
flutter test --machine integration_test | tojunit --output $junitReportFile
```

Remember that you need to have the emulator or driver running before executing the tests, please check more information here.

This command will also generate a JUnit report as we can see below.

junit-widget-report.xml

Integrating with Xray

As we saw in the above example, where we are producing JUnit reports with the result of the tests, it is now a matter of importing those results to your Jira instance. You can do this by simply submitting automation results to Xray through the REST API, by using one of the available CI/CD plugins (e.g. for Jenkins) or using the Jira interface to do so.

API

API

Once you have the report file available you can upload it to Xray through a request to the REST API endpoint for JUnit. To do that, follow the first step in the instructions in v1 or v2 (depending on your usage) to obtain the token we will be using in the subsequent requests.

Authentication

The request made will look like:

```
curl -H "Content-Type: application/json" -X POST --data '{ "client_id":
   "CLIENTID","client_secret": "CLIENTSECRET" }' https://xray.cloud.getxray.
   app/api/v1/authenticate
```

The response of this request will return the token to be used in the subsequent requests for authentication purposes.

JUnit XML results

Once you have the token we will use it in the API request with the definition of some common fields on the Test Execution, such as the target project, project version, etc.

```
curl -H "Content-Type: text/xml" -X POST -H "Authorization: Bearer
$token" --data @"junit-unit-report.xml" https://xray.cloud.getxray.app/api
/v2/import/execution/junit?projectKey=XT&testPlanKey=XT-582
```

With this command, you will create a new Test Execution in the referred Test Plan with a generic summary and one test with a summary based on the test name.

			4. Over resource	• · · · · ·
utorial-flutter				To Do 👻
🖗 Attach 🛛 Create subtask 🖉 Link issue 👻	🛉 Tests \cdots			Details
escription				Annianana
d a description				O Unassigned
				Assign to me
sts				Reporter
Tests Test Executions				😨 Cristiano Cunha
Add Tests Create Test Execution			View on board	Development
				🕼 Create branch 🛁
verall Execution Status		All Environments,	final status V	
				Labels
PASSED			TOTAL TESTS: 1	None
				Sprint
Only My issues Filters ~		10 🗸	Columns ~	XT Sprint 1
 Only My issues Filters Key Summary 	Assignee #Test Executions Dat	10 ¥	Columns ~ Actions	XT Sprint 1 Priority
	Assignee #Test Executions Dat	aset Latest Status ASSED	Columns ~ Actions	XT Sprint 1 Priority = Medium
V Only My Issues Fitters v Key Summary XT-588 Counter value should be incremented Prev 1 Next.	Assignee #Test Executions Dat	aset Latest Status PASSED	Columns ~ Actions JE ···· Total 1 issues	XT Sprint 1 Priority Medium Automation Rule executions
Conty My issues Filters Key Summary X1-588 Counter value should be incremented Prov 1 Next	Assignee #Test Executions Dat 1	aset Latest Status PASSED	Actions jE ···· Total 1 issues	XT Sprint 1 Priority = Medium Automation Ø Rule executions More fields Original estim
Cotyl My taxass Filters Kr 58 Counter value should be incremented Prev 1 Ned Add a comment.	Assignee #Test Executions Dat 1	aset Latest Status PASSED	Columna v Actions jii ···· Total 1 issues	XT Sprint 1 Priority Medium Automation GR Rule executions More fields Original estim Created 1 hour ago

Jira UI

Jira UI

Projects / 😲 Xray Tutorials / 💽	XT-584
Unit Test - Validate Co	punter
@ Attach 🛛 Create subtast	k 🔗 Link issue 👻 🦩 Test details 🚥
Description Add a description	
Test details	
	This test hasn't been executed, yet
	A test issue is a template that only contains the specification. To execute this test, you need to create a test run by starting an ad ho execution or by adding this test to an existing test execution issue.
	Execute in Very Very Very Very Very Very Very Very
	Existing test execution

~
2

Fill in the necessary fields and press "Create."

Create Test Execution	
Project	
Xray Tutorials	
Summary *	
Ad-hoc execution for XT-584	
Assignee	
Cristiano Cunha	~
Choose a user to assign the Test Executi Fix Version/s	2n
Select	~
Test Environment	
Select	~
C Execute Immediately	



Open the Test Execution and import the JUnit report.

Brians I D Yes Tavia I D Y.680		* * * * *
Ad-hoc execution for XT-584	To Do ~	
Ø Attach €) Create subtaak Ø Link Issue - O Tests ····	Octalia	Add flag
Description	Ansignes 🚳 Ortations Ou	Xray - Generate Cucumber Feature file
Tests	Reperter Cristiano C. Development Ir Create brand	Xray - Import Execution Results Xray - Document Generator
Add Texts (v)	Create comm	Connect Stack channel
Overall Execution Status	Labels None Priority = Medium	Convert to Subtask More
1 TOTAL TESTS: 1	Automation	Core
■ v Only My Text Runs Filters v Datumes v	Test Plans Open Test Plans Test Environments Open Test Environ	Find your field
Ratic: Kay: Summary: Text Type: Detailet: Molecci: Status: Actiona 1 XT-664 Unit Text - Validate Counter Generic 0 1000 1	More fields Original estimate, Time tracking, Ep	Actions menu
Par 1 Not Southand	Enanted 28 seconds ago Updated 28 seconds ago	Print Export XML Export Hord
Addivity School All Constants History Working Xiny-History Nessett Ent L7		NEW JIKA ISSUE NEW Take a tour
Add & konnert. Pre Na pres (k) to connert		Find out more See the old view

/	\frown	
(٨	
	4	
	\smile	

Choose the results file and press "Import."

Import Execution Results

Choose file No file chosen The file with the execution results for the Test Execution.

/		
(Б	
	0	
$\overline{\ }$	_	

The Test Execution is now updated with the test results imported.

Submit Cancel

	Xray Tutorials /	XT-589									
Ad-hoc	execution for	or XT-584									
@ Attach	Create subt	task 🔗 Link iss	ue v	O Tests	••••						
Description											
Add a descrip	tion										
Tests											
Add Tests	~									View on	board
Overall Ex	ecution Statu	s									
4											
PASSED										TOTAL TE	ISTS: 1
		ns Filters -							10 🗸	Columns	÷ ×
	Only My Test Ru										
■ ~ Rat	Only My Test Ru ik * Key : Su	ammary :		Tes	t Type :	Dataset	#Defects :	Status :			Actions
Rat	Only My Test Ru nk* Key: Su XT-588 Co	ummary :	be increme	Ter ited Ger	t Type : neric	Dataset	WDefects : 0	Status :		≣D	••••

Tests implemented will have a corresponding Test entity in Xray. Once results are uploaded, Test issues corresponding to the tests are auto-provisioned, unless they already exist.

Projects / 🚺 Xray Tutorials / 🧧 XT-588	
Counter value should be incremented	
Description	
Add a description	
Fest details	
🧮 Test details 🔯 Preconditions 💽 Test Sets 🛛 🛃 Test Plans 😰 Test Runs	
R - 1 R	Test Repository
Generic Y	
Definition	
.Users.cristianocunha.Documents.Projects.flutter.SF_flutter_demo.my_app.test.unit.Counter value should be incremented	
Activity	
Show: All Comments History Work log Xray History	Newest first 17
Add a comment	
Pro tip: press M to comment	

Xray uses a concatenation of the suite name and the test name as the the unique identifier for the test.

In Xray, results are stored in a Test Execution, usually a new one. The Test Execution contains a Test Run per each test that was executed.

Projects / 😲 Xray Tutorials / 🗈 XT-589	
Ad-hoc execution for XT-584	
Description Add a description	
Tests	
Add Tests 🗸	View on board
Overall Execution Status	
1 passed	TOTAL TESTS: 1
Only My Test Runs Filters	10 V Columns V
Rank* Key: Summary: Test Type	Dataset #Defects: Status: Actions
1 XT-588 Counter value should be incremented Generic	0 PASSED ED
Prev 1 Next	Total 1 issues

Detailed results, including logs and exceptions reported during execution of the test, can be seen on the execution screen details of each Test Run, accessible through the *Execution details*:

Projects / 🛐 Xray Tutorials / 🖪 XT-589			
Ad-hoc execution for XT-584			
🖉 Attach 🖾 Create subtask 🖉 Link issue 👻 🧕 Te	ests ····		
Description Add a description			
Tests			
Add Tests ~			View on board
Overall Execution Status			
1 _{passed}			TOTAL TESTS: 1
 Only My Test Runs Filters 			10 🛩 Columns 🗸
Rank to Key to Summary to	Test Type : Dataset	#Defects : Status :	Actions
1 XT-588 Counter value should be incremented	Generic	0 PASSED	ID
Prev 1 Next			Total 1 issues

As we can see here:



Tips

- after results are imported in Jira, Tests can be linked to existing requirements/user stories, so you can track the impact of their coverage.
- results from multiple builds can be linked to an existing Test Plan in order to facilitate the analysis of test result trends across builds.
- results can be associated with a Test Environment, in case you want to analyze coverage and test results by that environment later on. A Test Environment can be a testing stage (e.g. dev, staging, preprod, prod) or an identifier of the device/application used to interact with the system (e.g. browser, mobile OS).

References

- https://flutter.dev/
 https://docs.flutter.dev/testing
 https://github.com/TOPdesk/dart-junitreport
 https://developer.android.com/studio/intro

- Overview
 Prerequisites
 Application overview
 Implementing tests

 Unit Test
 Widget Test
 Integration Test

 Integrating with Xray

 API
 - - AuthenticationJUnit XML results

 - Jira UI
- TipsReferences