Flutter
Testing Code Samples

SUMMARY

A proposal to add unit testing to API documentation code samples.

Author: Greg Spencer (gspencergoog)
Go Link: flutter.dev/go/testing-code-samples
Created: 6/2021 / **Last updated:** 6/2021

OBJECTIVE

To create a way to create tests for API documentation code samples, so that we know that the samples have the functionality that we expect them to have, and that they function properly on all platforms. An additional objective would be to be able to enforce that a sample has a test as a presubmit test.

BACKGROUND

Currently, Flutter's API documentation code samples (e.g. this one) are analyzed and formatted, but no unit tests are run on them. There's not even a smoke test to make sure that they don't immediately crash. This has resulted in a few cases of samples that just don't run, as the code has changed underneath them.

The source for these samples is the API documentation itself. We store the samples in the `///` dartdoc comments in the code, surrounded by tags that look like `{@tool dartpad --template=material_scaffold}...{@end-tool}`. This location is part of what makes it difficult (currently impossible) to write unit tests for: they are only comments, not code in separate files. They are stored in the comments because it has been seen to be quite beneficial to have the samples next to the code that they illustrate: it puts them front and center when reading the comments in the code, and it makes it easier to search/replace terms in a file and also catch the ones in the sample code. The IDEs don't recognize them as code, however, so renaming

symbols doesn't carry into the samples.

Glossary

- **Code Sample** - A piece of code connected with the API documentation meant to illustrate a concept or mechanism in the codebase.

OVERVIEW

There are several proposals in this document, and one of the goals is to reduce the proposals to the one preferred proposal.

The presentation of the API documentation on the API documentation site would not change in any of these designs.

DETAILED DESIGN/DISCUSSION

Design One: Extract and Conquer (Currently Preferred)

This design takes the sample source out of the API documentation entirely, leaving only a placeholder in the docs that points to their location in the repo (a path), allowing us to run and write tests in the place where the source code is committed.

The down side of this is that the sample source is no longer visible in the API documentation comments. Searching in the same file for symbols won't search the sample code, although IDEs will now be able to see the sample code, so symbol renaming will work.

Pros:

- Only one source of truth for the sample source.
- Unit tests are written in the same way as other unit tests.
- Doesn't require any new presubmit checks or tools.
- Symbol renaming and searching in IDEs will work on the sample code too.
- Able to realize the goals of the Medium-Sized Sample Code proposal.
- Reuse of samples in multiple places is practical.

Cons:

- Sample source is no longer visible in API documentation comments.

Design Two: Separate But Equal

This proposal is to continue to have the comments be the source of truth for the code, but to *also* commit a copy of these samples to the repository. It means that we will need tooling which notices when a comment is changed, and updates the checked in copy. Alternatively, we could make the one in the separate file the source of truth, and running the tool just updates the comments.

This enables the ability to write tests alongside the checked in location for the sample code, and run those tests as part of continuous integration (CI) testing.

The down side is the need to run a command whenever you update a sample, and to have the CI system run a check to make sure you didn't modify a sample without running that command.

One variation of this design: The command could possibly sync both ways, allowing you to edit the code sample on disk and place the result back into the appropriate API doc comment, or vice versa.

Pros:

- Allows sample sources to stay in API documentation.
- Unit tests are written in the same way as other unit tests.
- Symbol renaming and searching in IDEs will work on the sample code too, provided the new sync tool is run afterwards to sync the comments.
- Able to realize the goals of the Medium-Sized Sample Code proposal.

Cons:

- Two sources of truth for sample sources.
- More complex presubmit checking is required.
- A tool is required to keep two copies in sync.
- Adds more procedure to checking something in.
- Adds to the learning cliff for new developers.
- Reuse of samples in multiple places is not practical.
- Large samples would use a lot of space in the comments.

Design Three: Testing In Absentia

This design leaves the sample source in the API documentation comments, and introduces a location in the repo where we can check in unit tests for each of the samples as separate files. To run the tests, a tool extracts the samples from the API documentation, merges them into a temporary project with the appropriate test file(s), and runs the tests.

Pros:

- Allows sample sources to stay in API documentation.
- Only one source of truth for the sample source.

Cons:

- Requires a tool and procedure for running sample tests.
- Requires a way of writing the sample tests that is different from the standard way, adding more to the learning cliff, and special casing in IDE plugins.
- Symbol renaming in IDEs will not work on the sample code (which is also a down side of the status quo).
- You couldn't run the test from within an IDE without making the IDE plugin aware of the extraction tool.

OPEN QUESTIONS

- Do we have good information about the usefulness of the samples being readable directly in the API documentation comments?

DOCUMENTATION PLAN

The mechanism for adding unit tests to code samples will be documented as part of the developer documentation in the Flutter wiki.